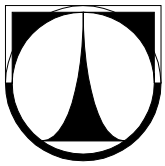
TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií



DIPLOMOVÁ PRÁCE

Liberec 2011 **Bc. Tomáš Hlava**

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Příspěvek k analýze možností testování software

Contribution to analysis of software testing possibilities

Diplomová práce

Autor: **Bc. Tomáš Hlava**

Vedoucí práce: Doc. Ing. David Vališ, Ph.D.

Konzultant: Doc. RNDr. Miroslav Koucký, CSc.

V Liberci 1. 1. 2011

ZDE VLOŽIT ORIGINÁLNÍ ZADÁNÍ

1. Úvod.
2. Terminologie ve spolehlivosti – se zaměřením na základní vlastnosti, ukazatele, životní cyklus a zkoušky spolehlivosti.
3. Životní cyklus softwaru – rozdíly oproti hardwaru, typy u softwaru, uplatnění, principy průběhu.
4. Zkoušky spolehlivosti/bezporuchovosti – typy zkoušek a jejich modely, formáty provedení, zásady provádění (rozsahy a omezující podmínky), hodnocení, rozdíly mezi zkoušením hardwaru a softwaru.
5. Návrh postupu pro testování softwaru v podmínkách vývoje nového softwaru.
6. Příklad aplikace testovacího postupu při vývoji nového softwaru.

Závěr

Cíle mé diplomové práce jsou:

- vypracovat/provést analýzu v problematice aktuálního pojetí spolehlivosti technických systémů.

- Popsat činnosti související se zkouškami bezporuchovosti hardwarových a softwarových systémů

- Navrhnout a popsat jednotlivé dílčí kroky nutné pro testování softwaru během vývojového cyklu

- Aplikovat navržený postup zkoušky spolehlivosti/bezporuchovosti na praktickém příkladu z praxe

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Poděkování

Chtěl bych vyjádřit poděkování Doc. Ing. Davidu Vališovi, Ph.D. za cenné připomínky a rady při vedení této diplomové práce. Rád bych poděkoval i Doc. RNDr. Miroslavu Kouckému, CSc. za užitečné konzultace. Poděkování patří také mým rodičům a přítelkyni za podporu během celého studia.

Abstrakt

Cílem diplomové práce je analýza současných postupů využívaných při testování software. Dále pak návrh postupu testování a nakonec realizace daného návrhu postupu testování software na projektu v praxi. Práce se snaží vysvětlit a přiblížit základní pojmy z oblasti testování. Zároveň představit procesy, které se v současné době využívají při zkouškách bezporuchovosti software. Přínos diplomové práce spočívá v návrhu postupu testování software a jeho následné realizaci na praktickém příkladu.

Práce je koncipována do dvou částí. První část se věnuje definici základních pojmů spolehlivosti, životnímu cyklu software a úvodu do problematiky zkoušek bezporuchovosti software. Druhá část práce obsahuje vlastní návrh postupu testování software. Návrh vychází z postupů definovaných v úvodní části této práce. Výsledný návrh je poté realizován na konkrétním příkladu postupu testování software v praxi. Výsledek realizace návrhu postupu testování softwaru v praxi je zaznamená v závěru druhé části této práce.

Cílem práce je informovat o současných možnostech testování software a zároveň představit jejich využití na konkrétním příkladu v praxi. Splnění tohoto cíle bylo dosaženo na základě studia ověřených materiálů, ale také na základě vlastních zkušeností ověřených v praxi na konkrétních případech.

Klíčová slova

testování softwaru, spolehlivost, bezporuchovost, návrh postupu testování, automatické testování

Abstract

The goal of this essay is the analysis of present-day processes used when the sofware is being tested. Also a draft of processes that are being used and eventually realization of mentioned draft of process of testing of the software in practice. This work is trying to explain and bring out basic terms used in testing. Additionally it presents processes of tests that confirm infallibility of the software. Benefits of this essay consist in the draft of process of testing of the software and its consecutive realization on practical example.

The essay is divided in two parts. The first part is focused on the definition of basic terms of infallibility, life cycle of the software and the introduction to issues of the tests of infallibility of the software. The second part contains the draft of process of testing of the software itself. The draft comes from processes defined in the first part of this essay. The essay is closed by a chapter about results of testing of this software.

The purpose of this essay is to inform about present-day posibilities of testing of the software and also to introduce their use on specific example in practice. Fulfillment of this goal was accomplished based on study of confirmed materials and it has been also supported by personal experiences approved in practice on specific examples.

Keywords

software testing, dependability, reliability, design of the software testing process, automated testing

OBSAH

Seznam obrázků

Seznam tabulek

1 Úvod

Téma této diplomové práce jsem si zvolil především na základě vlastních zkušeností v oblasti testování software. Mým prvořadým cílem bylo přispět k současné informovanosti o této problematice. Problematika možností testování softwaru je dle mého názoru stále podceňována. Přestože se této problematice zahraniční literatura věnuje poměrně obsáhle, kvalitních publikací v českém jazyce je velmi málo. Tato skutečnost jen odráží nezájem o toto téma. Přitom jde o obor, který může být pro firmy velkým přínosem, zejména co se týče úspory finančních prostředků. Z vlastních zkušeností mohu potvrdit, že firmy se v praxi potýkají z nedostatkem kvalifikovaných pracovníků v oblasti testování softwaru. V praxi tak nastávají situace, kdy si firmy musejí takovéto pracovníky samy „vychovat“. Což znamená investovat nemalé prostředky do jejich vzdělání. Spolehlivost produktů byla v poslední době zanedbávána na úkor zvýšení zisku a objemu výroby. Přitom investice do spolehlivosti produktu přináší také úspory během jeho životního cyklu.

V této práci bych rád představil současné možnosti zkoušek bezporuchovosti softwaru. Pro účely daného tématu a rozsahu diplomové práce jsem se snažil vybrat pouze nejpodstatnější části zmíněných oblastí, se kterými pracuji v průběhu práce. Práci lze rozdělit na část teoretickou a praktickou.

V úvodu práce se zmíním o základní terminologii ve spolehlivosti, zaměřím se pouze na oblasti, které využiji v dalších kapitolách. Po popsání životního produktu uvedu několik modelů životních cyklů softwaru. Dále se pokusím porovnat základní odlišnosti životních cyklů hardwaru a softwaru. V následující kapitole se budu věnovat zkouškám bezporuchovosti softwaru. Popíši několik základních kategorií testů a také úrovně, ve kterých se tyto testy provádějí. Na závěr této teoretické části provedu porovnání zkoušek bezporuchovosti mezi hardwarem a softwarem.

V praktické části této práce představím svůj vlastní návrh postupu testování a aplikuji jej na konkrétní příklad v praxi. Výsledkem bude konfrontace návrhu založeného jednak na informacích z předešlých kapitol a jednak na mých dosavadních zkušenostech z praxe. Návrh postupu testování není vytvořen pro určitý projekt, ale je sestaven obecně pro jakýkoliv postup testování softwaru. Aplikace toho návrhu je již velmi konkrétní.

Při testování software je třeba mimo jiné rozlišovat velikost (rozsah) projektu. Pro účely této práce, tuto problematiku zjednoduším na oblasti malých a velkých projektů. Malými projekty tedy chápejme zakázku na software, na jejíž realizaci se podílí tým čítající řádově desítky pracovníků. U takových projektů je také tým testerů omezen pouze na vedoucího testerů (test manažer) a několik jednotek testerů, kteří se fyzicky starají o vlastní testování software. Oproti tomu u velkých projektů celý realizační tým čítá stovky pracovníků. Tým testerů má k dispozici několik desítek zaměstnanců. Na přípravě testování se podílí také analytik a designér testů. Tvorbě automatizovaných testů se věnuje několik specialistů. Kromě samotného počtu pracovníků, kteří se podílí na životním cyklu produktu, samozřejmě existuje celá řada rozdílů mezi velkými a malými projekty. Za všechny jmenujme např. rozsah a typy dokumentace, softwarové nástroje apod.

Zejména v teoretické části této práce využívám ve vztahu k bezporuchovosti softwaru pojem zkouška. Jelikož se však v praxi, především ve spojení s vývojem softwaru využívá více pojem test, v druhé polovině práce záměrně využívám toho (mě osobně bližšího) pojmu.

V práci také často využívám ve spojitosti se softwarem pojmy chyba a porucha. Tyto pojmy nelze zaměňovat. Norma ČSN EN 61508-4 Funkční bezpečnost elektrických/elektronických/programovatelných elektronických systémů souvisejících s bezpečností - Část 4: Definice a zkratky [1] definuje chybu jako: *Nesoulad mezi počítanou, pozorovanou nebo měřenou hodnotou nebo podmínkou skutečnou definovanou nebo teoreticky správnou hodnotou nebo podmínkou.* Porucha je naopak definována jako: *Ukončení schopnosti funkční jednotky plnit požadovanou funkci nebo provoz funkční jednotky jiným než požadovaným způsobem*. Funkční jednotka je pak v normě popsána jako: *Entita hardwaru nebo softwaru nebo obou, schopná plnit stanovaný účel*. Z výše uvedených definic je tedy zřejmé, že pojem chyba bude v této práci v souvislosti s testování softwaru využíván v případě nesouladu skutečného chování softwaru a požadavků na jeho chování. Naopak pojem porucha bude využit v případě ukončení schopnosti softwaru plnit požadovanou funkci.

2 Analýza aktuálního pojetí spolehlivosti

Než se začneme zabývat možnostmi testování software, je nutné si definovat pojmy, které jsou využívány v dalších kapitolách. Tato úvodní kapitola se věnuje především definici spolehlivosti, jejích vlastností a ukazatelů. Druhá část kapitoly je věnována popisu spolehlivosti během životního cyklu produktu, konkrétně pak softwaru hardwaru.

Výše popsaným oblastem je v této kapitole věnován prostor také proto, abychom při provádění zkoušek mohli spolehlivost software pochopit v širším pojetí.

2.1 Definice spolehlivosti

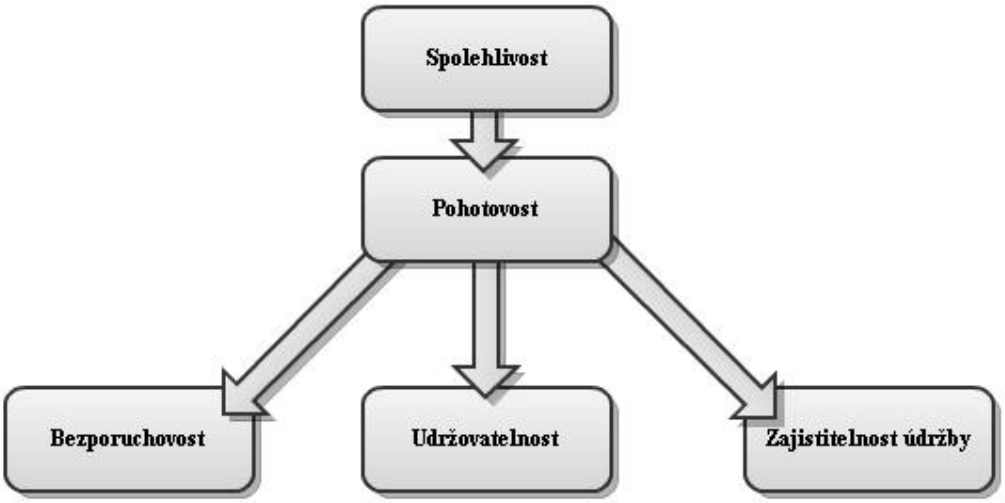
Ve všech technických oborech se používají speciální odborné termíny, které jsou přesně definovány v základních názvoslovných normách. Výjimkou není ani obor spolehlivosti. Pokud by nebyla všechna názvosloví v normách definována, docházelo by k záměně termínů a jejich nesprávnému výkladu. Základní názvosloví pro obor spolehlivosti jsou definována v:

ČSN IEC 50(191):1993 (01 0102) 80 Mezinárodní elektrotechnický slovník – Kapitola 191: Spolehlivost a jakost služeb (k této normě byly vydány změny Z1:2003 a Z2:2003).

Tato norma obsahuje definice pojmů a ekvivalenty termínů z tohoto oboru v 11 jazycích. Slovensky, česky a anglicky je definováno několik set hesel. Je opatřena abecedními rejstříky definovaných termínů ve všech 11 jazycích. Je všeobecně využitelná jako základní názvoslovná norma pro management spolehlivosti zařízení a služeb pro energetiku. Je nezbytná pro porozumění a jednotné používání základních termínů z oboru spolehlivosti.

Podle [2] je spolehlivost definována jako **„Souhrnný termín používaný pro popis pohotovosti a činitelů, které ji ovlivňují: bezporuchovost, udržovatelnost a zajištěnost údržby (používá se pouze pro obecný nekvantitativní popis).“**

Spolehlivost je komplexní vlastnost, která může zahrnovat např. bezporuchovost, životnost, udržovatelnost, skladovatelnost. Schéma spolehlivosti v užším pojetí je znázorněno na obrázku obrázek 1:.



Obrázek 1: Spolehlivost v užším pojetí – dle [2]

Ve vztahu k informačním technologiím se využívá norma:

ČSN ISO/IEC 2382-14:1999 (36 9001) Informační technologie – Slovník – Část 14: Bezporuchovost, udržovatelnost a pohotovost [3].

Norma obsahuje názvosloví z oboru spolehlivosti používané v oboru informačních technologií. Česky a anglicky je definováno cca 42 hesel. V normě je uveden český, anglický a francouzský abecední rejstřík. Je všeobecně využitelná jako základní názvoslovná norma pro management spolehlivosti zařízení a služeb pro energetiku v oboru informačních technologií. Je nezbytná pro porozumění a jednotné používání základních termínů z oboru spolehlivosti ve vztahu k informačním technologiím.

2.2 Základní pojmy spolehlivosti

Pro správné pochopení souvislostí v této práci, je nutné si nejprve vymezit základní použité pojmy a termíny. Tak jako v ostatních oblastech, tak i v oblasti spolehlivosti, je správnému vymezení pojmů věnována velká pozornost i na mezinárodní úrovni (viz Mezinárodní normy v předcházející kapitole 2.1).

2.2.1 Vybrané sledované objekty ve spolehlivosti

Objekt je (dle [2]) jakákoliv součást, zařízení, část systému, funkční jednotka, přístroj, systém, s kterým je možné se individuálně zabývat. Je-li objekt na dané úrovni

dále nedělitelný, nazýváme jej prvkem. Systém je souhrn několika vzájemně souvisejících nebo vzájemně působících prvků. Objekt může obsahovat jak hardware, tak i software dokonce do něho mohou být zahrnuti i lidé.

Sledované objekty ve spolehlivosti jsou nečastěji výrobky nebo služby. Hodnocení spolehlivosti se vztahuje vždy k plnění požadované funkce.

Dále jsou v následujícím textu ve vztahu k výrobku (službě) rozlišovány tři hlavní subjekty, dodavatel, zákazník a uživatel.

Dodavatel je právnická či fyzická osoba, která svými činnostmi nebo procesy vytváří výrobek. Ten to výrobek následně poskytuje zákazníkovi.

Zákazník je subjekt (právnická či fyzická osoba), který je příjemcem výrobku poskytnutého dodavatelem.

Uživatel je subjekt (právnická či fyzická osoba), který využívá výrobek (službu) k plnění požadovaných funkcí. Předpokládá se, že uživatel má k výrobku jistý vlastnický vztah a že hradí (nebo se podílí na hrazení) náklady spojené s vlastnictvím výrobku.

V této práci bude náš sledovaný objekt software. Zákazníkem je subjekt, který si od nás (dodavatele) objednal výrobu software. Uživatelem pak chápejme osoby, jež budou tento software využívat ke každodenní práci.

Udržovatelnost objektu

Z pohledu udržovatelnosti lze objekty rozdělit na:

Opravované objekty – Jedná se o objekt, který je po poruše způsobilý k opravě. Po opravě je obnovena jeho funkce.

Neopravované objekty – U těchto objektů nemůže být obnova funkce zajištěna opravou vadného/porouchaného prvku. Porouchaný/vadný prvek může být nahrazen novým bez poruchy.

Pro účely této práce budeme na testovaný software nahlížet jako na opravovaný objekt. Po poruše tedy bude skutečně opraven.

2.2.2 Vlastnosti objektu

Dle [2] je **spolehlivost** chápána úžeji jako termín pro popis **pohotovosti** a činitelů, které ji ovlivňují: bezporuchovost, udržovatelnost a zajištěnost údržby. U objektu sledujeme především jeho spolehlivost. Pojem spolehlivost je užíván jako obecný termín a nelze jej kvantifikovat žádnou hodnotou. Její jednotlivé dílčí činitele (např. pohotovost apod.) však již hodnotit možné je.

Pohotovost

Schopnost objektu být ve stavu schopném plnit požadovanou funkci v daných podmínkách, v daném časovém okamžiku nebo v daném časovém intervalu, za předpokladu, že jsou zajištěny požadované vnější prostředky (údržby). Pohotovost je hlavní vlastnost spolehlivosti, zahrnující bezporuchovost, udržovatelnost a zjistitelnost údržby.

Bezporuchovost

Schopnost objektu plnit požadovanou funkci v daných podmínkách a daném časovém intervalu.

Udržovatelnost

Schopnost objektu v daných podmínkách používání setrvat ve stavu, nebo se vrátit do stavu, v němž může plnit požadovanou funkci, jestliže se údržba provádí v daných podmínkách a používají se stanovené postupy a prostředky.

Zajištěnost údržby

Schopnost organizace poskytující údržbářské služby zajišťovat dle požadavků v daných podmínkách (vztahují se jak na vlastní objekt, tak na podmínky používání i údržby) prostředky potřebné pro údržbu podle dané koncepce údržby.

Životnost

Schopnost objektu plnit požadovanou funkci v daných podmínkách používání a údržby do mezního stavu, který lze charakterizovat ukončením užitečného života, ekonomickou nebo technickou nevhodností, či jinými závažnými důvody.

Z výše uvedených vlastností objektu je pro popis provádění zkoušek bezporuchovosti software v této práci zásadní zejména bezporuchovost. Samotnou

bezporuchovostí software se budeme zabývat prakticky v celé této práci. Zkouškám bezporuchovosti je pak věnována kapitola 3.

2.2.3 Vybrané ukazatele vlastností spolehlivosti

Jak již bylo uvedeno v úvodu této kapitoly, spolehlivost má své dílčí subvlastnosti. Tyto subvlastnosti mají své ukazatele. Proto pokud mluvíme o ukazatelích ve vztahu ke spolehlivosti, musíme vždy uvést, ke které vlastnosti jsou vztaženy. Ukazatele jsou vyjadřovány například pomocí matematických funkcí nebo fyzikálních veličin.

Vlastnosti objektů ve spolehlivosti již byly uvedeny v kapitole 2.2.2. K jejich kvantifikaci se využívá ukazatelů.

Ukazateli se v pravděpodobnostním pojetí spolehlivosti rozumí funkce nebo hodnota, používaná pro popis náhodné proměnné nebo náhodného procesu. Obecně jsou ve vztahu ke spolehlivosti ukazatele chápány jako nástroje umožňující popis stochastických jevů a procesů, které charakterizují spolehlivost objektu. Všechny definice ukazatelů (uvedených subvlastností spolehlivosti) jsou uvedeny v [2].

Pro potřeby této práce využijeme jen několik ukazatelů, které jsou uvedeny níže v tabulce Tabulka 1:.

Tabulka 1: Vybrané ukazatele vlastností spolehlivosti

Sledovaná vlastnost	**Název ukazatele**	**Označení**	**Anglický název**
Bezporuchovost	Střední doba provozu mezi poruchami	$\bar{t}$	Mean Time Before Rate - MTBR
Udržovatelnost a zajištěnost údržby	Střední doba opravy	$\bar{t}_{oo}$	Mean Repair Time - MRT

2.3 Spolehlivost během životního cyklu produktu

Životní cyklus produktu

Životní cyklus produktu je časový interval od stanovení koncepce produktu po jeho vypořádání (likvidaci) [4]. Tvoří jej celkem šest etap. Tyto etapy jsou zobrazeny na

obrázku Obrázek 2:. Níže jsou jednotlivá období stručně charakterizována, tak jak jsou definována v [5].

Obrázek 2: Etapy životního cyklu produktu dle [4]

Etapa koncepce a stanovení požadavků

V etapě koncepce a stanovení požadavků se provádí sběr požadavků na nový produkt a jejich následné zdokumentování. Během této etapy může být uskutečněn průzkum trhu. Vyhotovena je analýza koncepce a návrhu produktu. Připraví se specifikace požadavků na produkt. Jsou stanoveny cíle spolehlivosti pro následující etapu. Vytváří se základy spolehlivosti produktu. Rozhodnutí během tohoto období, mají největší dopad na finální produkt.

Etapa návrhu a vývoje

Během etapy návrhu a vývoje se vytváří podrobná dokumentace a samotný vývoj produktu. Provádí se technické práce na návrhu včetně činností týkajících se

zajištění bezporuchovosti, udržovatelnosti a ochrany proti vlivům prostředí. Probíhá sestavení a analýza predikce spolehlivosti vycházející z použitých řešení. Je zhotoven prototyp. Pomocí zkoušek je ověřeno plnění stanovených cílů spolehlivosti.

Etapy výroby

Etapou výroby se nazývá období, kdy je zhotoven produkt na základě připravené dokumentace. Z pohledu spolehlivosti je velmi důležité dodržení všech parametrů stanovených v dokumentaci. Prováděny jsou schvalovací typové zkoušky (kvalifikační).

Etapa instalace

V etapě instalace je produkt instalován na místo svého provozu. Charakter této etapy je závislý na typu zákazníka a charakteru produktu. Produkt je uveden do zkušebního provozu. Integrovány jsou funkce pro zajištěnost údržby. Před finálním předáním musí být produkt otestován v reálném prostředí.

Etapa provozu a údržby

Etapa provozu a údržby je z časového hlediska nejdelším obdobím životního cyklu produktu. V této etapě je produkt využíván k zamýšlenému účelu. Je nutné zajistit technologii údržby a oprav. Provádí se školení obslužného personálu. Provozní spolehlivost je v této etapě ovlivněna určením optimálních intervalů pro provádění preventivní údržby. Konec užitečného života produktu nastává, je-li jeho provoz nehospodárný v důsledku zvýšených nákladů na zajištěnost údržby nebo vlivem jiných faktorů.

V etapě provozu a údržby se někdy vyskytuje sub-etapa **modernizace**. Zde lze zdokonalováním produktu zvýšit jeho kvalitu. Předtím je však nutné zhodnotit možné přínosy modernizace a jejich ostatní dopady (náklady apod.). Dále pak stanovit minimální hodnoty parametrů spolehlivosti pro nový (zdokonalený) produkt.

Etapa vypořádání (likvidace)

V poslední etapě životního cyklu je ukončeno používání produktu. Produkt je odstaven, fyzicky zlikvidován či demontován, záleží na jeho charakteru. Proběhne oficiální ukončení provozu. Je možné provést zkoušky a analýzy opotřebení, stanovit tak zbytkovou životnost. Tyto výsledky pak slouží ke zlepšení úrovně spolehlivosti nového produktu.

Výše zmíněné členění etap je možné použít i pro obecný životní cyklus software. Obsahuje všechny hlavní fáze a posloupnost tak, jak je využívána v praxi. Jednotlivé etapy obsahují různé procesy a činnosti, ty jsou však pro softwarový produkt specifické. Procesy a činnosti jednotlivých etap popisují modely životního cyklu software. V další kapitole jsou představeny některé nejznámější modely.

Spolehlivost během životního cyklu produktu

Dostatečná pozornost na spolehlivost musí být věnována po celou dobu životního cyklu produktu. Dle [6] během etapy návrhu a vývoje a také etapy výroby vzniká tzv. Inherentní spolehlivost, tj. *Spolehlivost „vložená“ do objektu v průběhu jeho návrhu a výroby. Nezahrnuje zhoršující vlivy provozních podmínek, podmínek prostředí, způsobů údržby, lidského faktoru apod.* Do etapy návrhu a vývoje spadá také tzv. odhadovaná (predikovaná) spolehlivost, tj. *Spolehlivost, která je výsledkem výpočtů, analýz a prognóz spolehlivosti projektovaného objektu. Je tedy výsledkem použitých metod odhadu, vstupních informací o spolehlivosti prvků, použitého výpočtového modelu spolehlivosti systému, schopností a možností analytika provádějícího odhad apod.* Správné stanovení metod pro zkoušení spolehlivosti má v této etapě zásadní dopad na pravděpodobnost bezporuchového provozu produktu.

V dalších etapách je nutný sběr a analýza dat o spolehlivosti. Údaje o spolehlivosti se získávají zkouškami bezporuchovosti. Podrobněji se zkouškami zabývám v kapitole 3. Získaná data musí být zpracována a vyhodnocena. Jen tak lze reagovat na vyvstalé problémy.

2.4 Software a jeho životní cyklus – rozdíly v životním cyklu v porovnání s hardware

Tato kapitola popisuje životní cyklus softwaru. Představuje základní modely těchto cyklů včetně jejich výhod a nevýhod pro použití v praxi. Na konci kapitoly jsou zmíněny odlišnosti životních cyklů software a hardware.

2.4.1 Životní cyklus software

Norma ISO 12207 - Informační technologie - Procesy v životním cyklu software [7] vytváří obecný rámec pro procesy životního cyklu softwaru s dobře definovanou terminologií, na níž se může softwarový průmysl odvolávat. Existují i další normy

popisující životní cyklus softwaru (např. ČSN EN 62304 - Software lékařských prostředků - Procesy v životním cyklu softwaru). Pro potřeby této práce (zejména pak této kapitoly) jsem se však rozhodl vycházet z výše zmíněné normy [7]. Zmíněná norma, dle mého názoru, dostatečně popisuje životní cyklus softwaru od stanovení koncepce až po vyřazení softwaru. Norma je navržena tak, aby mohla být využitelná pro řízení projektů informačních technologií. Dle této normy je životní cyklus definován jako: *Vývoj systému, produktu, služby, projektu či jiný entity vyráběných člověkem od jejího vzniku až do zániku.*

Model životního cyklu pak norma definuje jako: *rámec procesů a činností zabývající se životním cyklem, který může být uspořádán do fází, které rovněž slouží jako společný referenční rámec pro komunikaci a porozumění.*

Proces životního cyklu podle normy ISO 12207 [7]

- Primární procesy
    - Smluvní pohled: proces pořízení, proces dodávky
    - Inženýrsky pohled: proces vývoje, proces údržby
    - Provozní pohled: proces provozu
- Podpůrné procesy
    - Dokumentace
    - Konfigurační řízení
    - Pohled jakosti: zajištění jakosti, verifikace, validace, kontrolní dny, audit
    - Řešení problémů
- Organizační procesy
    - Manažerský pohled: proces řízení
    - Proces infrastruktury
    - Proces zdokonalování
- Proces školení

Norma ISO 12207 [7] nenařizuje dělení životního cyklu do určitých fází. Doporučuje dělení podle procesů. Každý proces představuje skupinu činností a úkolu,

které je nutné provést k dosažení úspěšného výsledku. Pro konkrétní případy je tak možné využít jen některé vybrané procesy. V některých případech je výhodnější se zaměřit spíše na samotné procesy než na obecné fáze cyklu. Zvláště pak pokud s nimi například nemáme mnoho zkušeností. V praxi je obvykle vybrán model životního cyklu. Ten již obsahuje soustavu procesů, úkolů a činností, jež pokrývají celý životní cyklus software.

V rámci vývoje softwaru lze nalézt mnoho modelů životního cyklu. Vybírat se mezi nimi musí velmi uvážlivě. Aby zvolená metodika mohla být úspěšně použita, musejí být splněny podmínky pro její realizaci.

2.4.2 Modely životního cyklu software

Model životního cyklu popisuje vzájemné vztahy mezi fázemi životního cyklu softwaru. Každý model obsahuje vlastní metodiku jak zajistit dostatečnou kvalitu produktu. V tomto kontextu často bývá pojem model a metodika zaměňován.

Výběr vhodného modelu životního cyklu je klíčový pro úspěch celého projektu. O důležitosti správného výběru se zmiňuje James Chapman [8]: *„složitým problémem při výběru a dodržování metodiky je činit tak s rozumem – poskytnout dostatek procesních disciplín k zajištění kvality vedoucí k obchodnímu úspěchu, ale zároveň se vyvarovat kroků, které představují ztrátu času, snižují produktivitu, demoralizují vývojáře a vytvářejí nepotřebnou administrativu.“* V současné době je modelů životního cyklu software velké množství. Většina z nich ovšem vychází z původních definic vodopádového nebo spirálového modelu (viz níže). Mnoho firem však nyní začíná využívat modelu RUP (*Rational Unified Process*), který je popsán na konci této podkapitoly.

Vodopádový životní cyklus (The waterfall Life cycle)

Nejstarší model životního cyklu software se nazývá Vodopádový. Jeho pojmenování vychází z přirovnání posloupnosti jednotlivých fází k protékání vody vodopádem, jak je znázorněno na obrázku Obrázek 3:. Winston W. Royce jej poprvé definoval ve svém článku „Managing the Development of Large Software System“ v roce 1970 [9]. Model byl vyvinut s cílem lépe se vyrovnat s rostoucí složitostí produktů leteckému průmyslu. Royce se ve svém článku zmiňuje o sedmi základních fázích:

- Systémové požadavky (System requirements)
- Softwarové požadavky (Software requirements)
- Analýza (Analysis)
- Návrh programu (Program design)
- Implementace (Coding)
- Testování (Testing)
- Provoz (Operations)

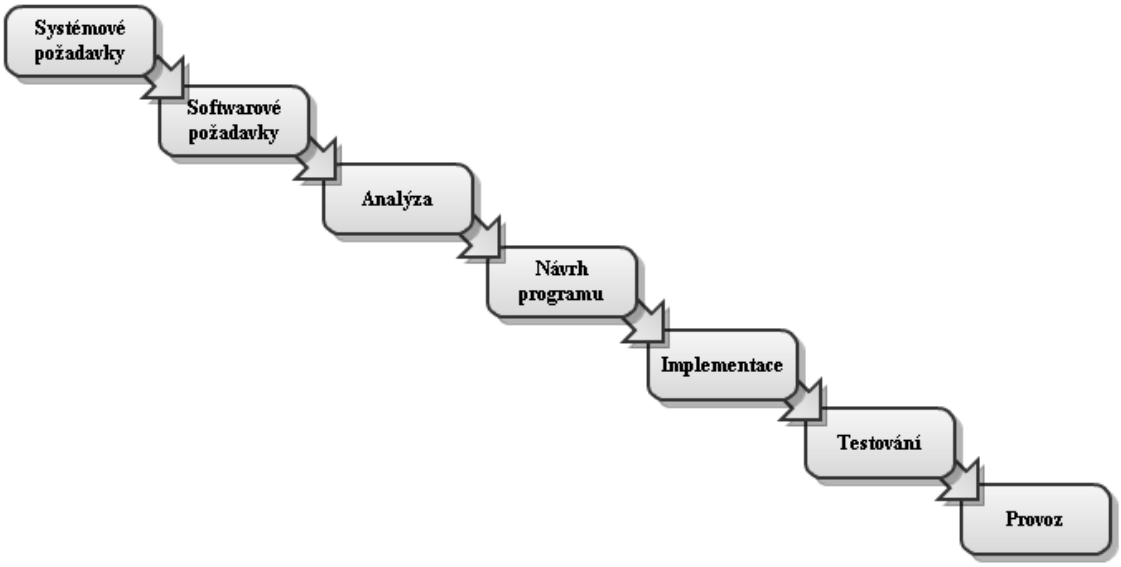
Základní myšlenka vodopádového modelu, vychází ze sekvenčního přístupu k jednotlivým fázím. Model je charakteristický tím, že vstoupit do další fáze mohu až tehdy, pokud je předchozí kompletně dokončena a uzavřena.

S úspěchem lze model využít tehdy, pokud je počátečním fázím věnováno dostatek času. Jen tak lze přijít k vyšším úsporám v pozdějších fázích životního cyklu. Odhalení a odstranění chyby, která je objevena v počátcích životního cyklu (např. v analýze), je mnohem levnější než kdybychom tutéž chybu opravovali později (např. při testování) [10]. V průběhu realizace projektu může nastat situace, kdy návrh programu nelze z nějakého důvodu implementovat. Pokud je tato skutečnost zjištěna již ve fázi návrhu, je přepracování návrhu mnohem jednodušší, než kdyby byla chyba v návrhu objevena v dalších fázích. Chceme-li tedy postupovat přesně podle vodopádového modelu, musíme si být na konci každé fáze maximálně jisti její validací a kompletností. Jen tak lze úspěšně zahájit následující fázi cyklu.

Hlavní nevýhodou Vodopádového modelu je fakt, že v praxi, zejména u rozsáhlejších projektů, nelze prakticky dokončit jednu fázi a zahájit další, beztoho aniž bych se k ní v budoucnu opět vrátil. Požadavky klientů se mohou v průběhu realizace programu měnit. Klient může vznášet další požadavky po zhlédnutí prototypu. V takovýchto případech je nutné zapracovat změny do specifikace a veškeré dokumentace.

Mezi další nevýhody patří prakticky nulová možnost reagovat v průběhu vývojového cyklu na pozměňovací požadavky zákazníka. Finální verze aplikace je zákazníkovi předána až v době, kdy se již nelze vrátit do fází oprav a úprav. Toto riziko může vést k neúspěchu celého projektu. Z pohledu testování je pak velmi nešťastné,

když fáze testování nastává až po samotné implementaci, tedy ve chvíli kdy je aplikace téměř připravena na předání zákazníkovi. Opravy chyb v této fázi jsou časově mnohem náročnější, než např. ve fázi návrhu. Pokud se takto objeví chyba v analýze, může vyústit až v kompletní přepracování projektu.



Obrázek 3: Vodopádový model [9]

Vodopádový model je jednoduchý a snadno pochopitelný. Jeho požívání vyžaduje, aby implementace postupovala přesně podle prověřeného návrhu. Tím je zajištěna snadná integrace systému. Zpravidla bývá využit u menších projektů, kde se nepočítá s pozdějšími změnami funkcionalit či vlastností celého výsledného programu.

Z původního modelu, který představil Royce, v současné době vychází spousta modifikací. Tyto modifikace se snaží vyřešit jeho nedostatky. Na druhou stranu, spousta modelů životního cyklu software, bude mít určitou podobnost s vodopádovým. Většina soudobých modelů totiž obsahuje fáze, jejichž návaznost se může podobat těm ze schématu vodopádového modelu. Jako příklady životních cyklů, vycházejících z Royceova původního modelu, lze uvést např. V – životní cyklus (*The V Life Cycle*) nebo Sashimi apod.

Vodopádový model, tak jak jej původně představil Royce, se dnes díky výše zmíněným nedostatkům v praxi příliš nepoužívá. Zejména pak plánování procesu testování až téměř na konec celého modelu, neumožňuje v dnešní době vodopádový model využít v praxi. Dnes se tento model využívá především k výukovým účelům.

Snadno se na něm velmi jednoduše demonstruje model životního cyklu software. Přes to všechno jej lze s určitými úpravami použít u drobnějších projektů, kde například není dostatečný časový nebo finanční prostor pro plánování (což ovšem samo o sobě není příliš šťastné a může se to negativně projevit na pravděpodobnosti bezporuchovosti provozu softwaru).

Spirálový model

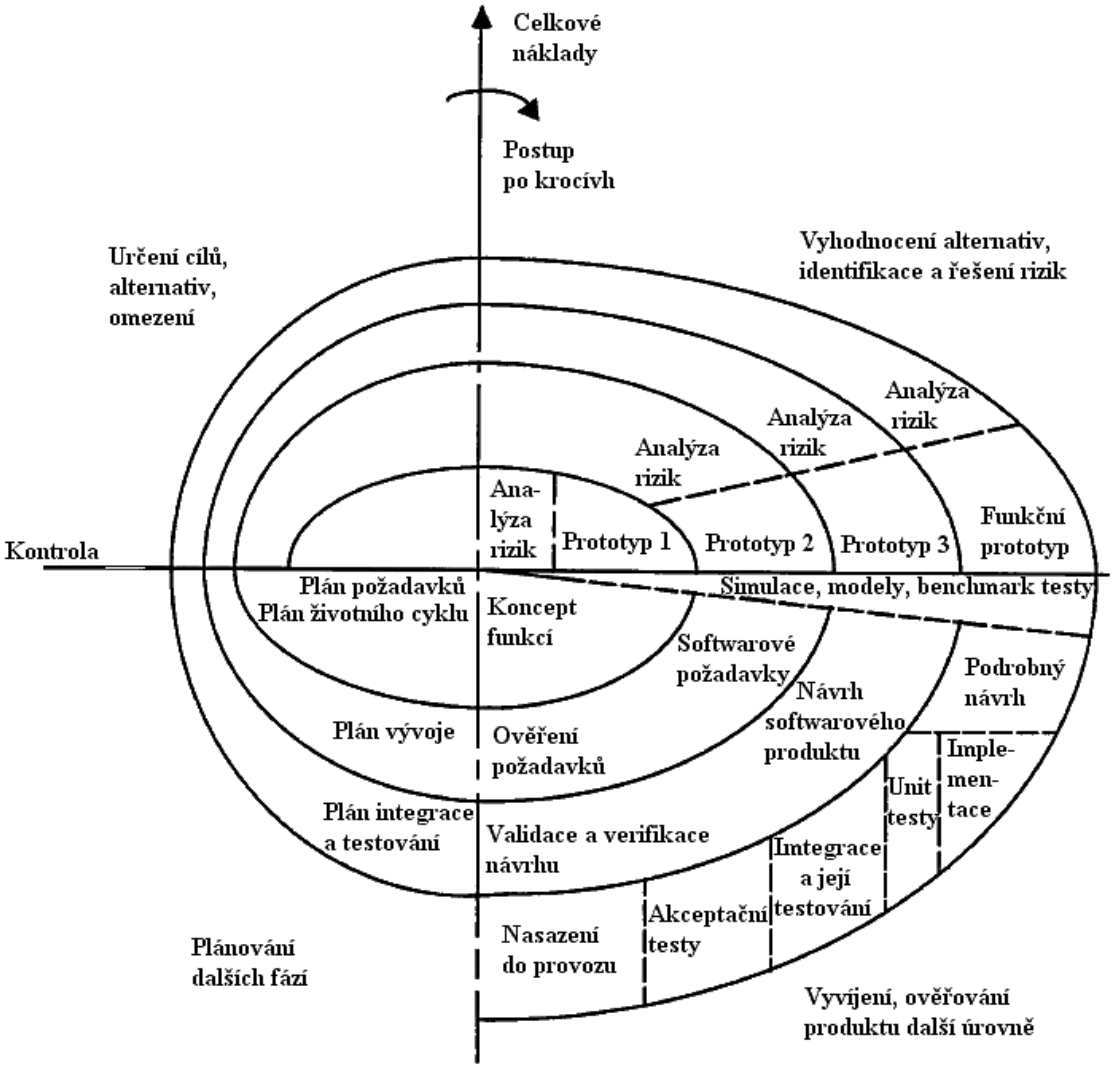
Barry Boehm poprvé definoval Spirálový model ve svém článku v roce 1986 (A Spiral Model of Software Development and Enhancement) [11]. Tento model velmi dobře pokrývá největší nedostatky vodopádového modelu [12]. Spirálový model náleží do skupiny tzv. přístupů řízených riziky. To znamená, že postup do další fáze závisí na důsledně provedené analýze všech rizik a možných problémů. Rizika lze v kontextu spirálového modelu chápat v nejobecnějším smyslu (mohou tak zahrnovat např. i vztah k legislativě či marketingu). Model je založen na iterativním přístupu a především zavádí opakovanou analýzu všech rizik. Lépe se tak vyrovnává s pozdější úpravou požadavků. Proto je model vhodný pro větší projekty. Model probíhá v několika krocích, které se neustále opakují, dokud není produkt hotov. Hlavní myšlenkou je zde navazování nových částí na již důkladně prověřený základ. Z počátku se vývoj provádí na základě hrubé specifikace požadavků, v pozdějších fázích je tato specifikace i po konzultaci se zákazníkem postupně upřesňována.

Celý životní cyklus podle Spirálového modelu je rozdělen do čtyř hlavních částí:

- Určení cílů, alternativ, omezení (Determine objectives, alternatives, constraints)
- Vyhodnocení alternativ, identifikace a řešení rizik (Evaluate alternatives, identify, resolve risks)
- Vývoj a verifikace další úrovně produktu (Develop, verify next-level product)
- Plánování následujících fází (Plan next phases)

Po každé fázi následuje testování, hodnocení a předání dílčích výsledků. Produkt je tedy testován pravidelně a to již od raných fází. S tímto přístupem je vhodné využít automatizovaných testů. Dle aktuální verze je pouze nutné upravit testovací případy a testovací scénáře. Aplikaci je možno testovat po částech. Díky pravidelnému a včasnému testování dochází k včasnému odhalení chyb. Velký počet chyb v počátcích

vývoje může mít za následek úpravu analýzy. Spirálový model, jak ho původně publikoval Barry Boehm [11], vidíme na obrázku Obrázek 4:.



Obrázek 4: Spirálový model – převzato z [12]

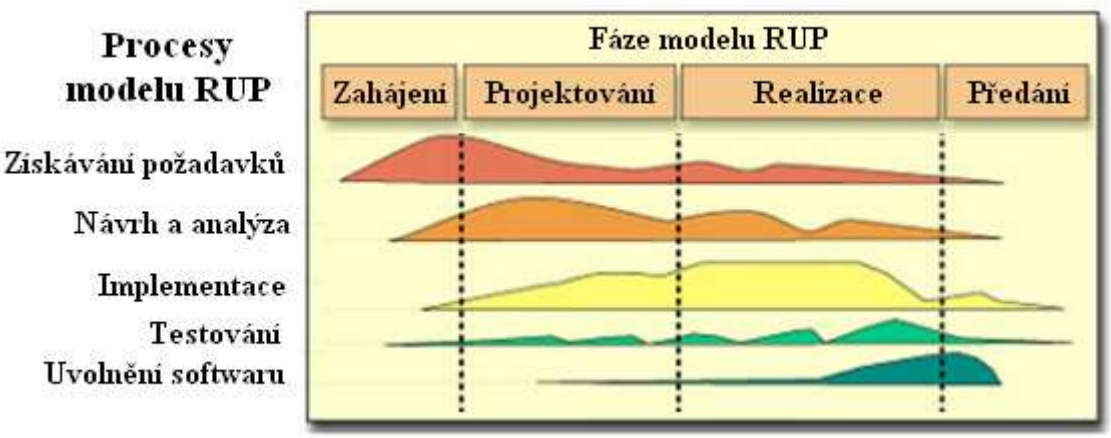
RUP (Rational Unified Process [13][14])

Rational Unified Process je objektově orientovaný iterativní přístup k životnímu cyklu software. Metodiku původně vytvořila společnost Rational Software Corporation (nyní samostatná divize IBM [15]). RUP náleží do skupiny tzv. přístupů řízených případy použití (use-casedriven approach), což znamená, že jako základní element je chápán případ použití definovaný jako posloupnost akcí prováděných systémem či uvnitř systému, která poskytuje určitou hodnotu uživateli systému (v nejobecnějším

smyslu [16]). Pro modelování procesů se využívá prostředků jazyka UML (*Unifed Modeling Language*)[1].

Hlavní myšlenka modelu je zjednodušena pro potřeby testování softwaru a zobrazena na obrázku Obrázek 5:. Na vodorovné ose jsou zaneseny fáze metodiky RUP. Fáze v tomto případě neodpovídají etapám životního cyklu produktu (tak jak byly uvedeny v kapitole 2.3), ale dynamickému rozdělení metodiky RUP (jejich popis je uveden níže v této kapitole). V jednotlivých fázích se někdy uvádějí tzv. iterace, což jsou podmnožiny jednotlivých fází a představující jejích milníky. Svislá osa obsahuje vybrané procesy (které mají přímý vliv na testování softwaru v metodice) včetně důrazu, jaký je na ně kladen během metodiky RUP. Pojem „důraz“ v tomto kontextu chápejme jako objem pracovních kapacit rezervovaných v danou chvíli na konkrétní činnost. Ilustrace znázorňuje, jaký důraz je na daný proces v průběhu životního cyklu kladen. Z obrázku je patrné, že na testování je kladen důraz během celého modelu. Nejvíce však těsně před předáním produktu zákazníkovy, tedy ke konci implementace.

RUP obsahuje celkem čtyři základní fáze. Každá fáze obsahuje několik dalších iterací. Před započetím nové iterace musí být splněna dříve definovaná kritéria předchozí iterace. Fáze zahájení (inception) definuje účel, rozsah projektu a jeho obchodní kontext. Ve fázi projektování (elaboration) je potřeba analyzovat požadavky zákazníka, celého projektu a definovat základy architektury. Realizační fáze (construction) je nejdelší probíhá zde tvorba zdrojových kódů. V poslední fázi předání (transition) může být projekt předán zákazníkovi nebo do dalšího cyklu.



[1] Unifed Modeling Language je do češtiny překládáno jako Sjednocený modelovací jazyk. Dle [17] je UML druh grafické notace podporovaný nezávislým meta-modelem, který umožňuje popisovat a navrhovat softwarové systémy, konkrétně systémy budované využitím objektově orientované metodiky.

Obrázek 5: Důraz na vybrané disciplíny během metodiky RUP

RUP také definuje 6 základních procedur, používaných při vývoji software:

1. Iterativní vývoj software
2. Správa požadavků
3. Architektura založená na komponentách
4. Vizuální modelování
5. Ověřování kvality software
6. Řízení změn software

Oblasti testování se metodika RUP věnuje velmi rozsáhle. Přesně definuje několik rolí a aktivit, které se mají testování věnovat. Testování prochází mnoha fázemi, proto lze testy neustále rozvíjet. Po ukončení procesu testování je vyhotovena zpráva, která obsahuje mimo jiné informace o tom, které části je nutné pro další testování pozměnit.

Z výše uvedených modelů životního cyklu software, se osobně přikláním k využívání modelu RUP. Nejen z pohledu testování software vykazuje tento model výborné výsledky na aplikovaných projektech. Detailní propracovaností a návazností jednotlivých etap předchází mnoha nepříjemnostem, které se mohou v průběhu životního cyklu software vyskytnout. Na druhou stranu u menších projektů, lze jen těžko aplikovat tento model. V tom případě je podle mne nejjednodušší využít vodopádový model nebo některou z jeho „mutací“.

2.4.3 Odlišnosti životního cyklu hardware a software

V kapitole 2.3 je popsán životní cyklus produktu. Z něj pochopitelně vychází i životní cyklus hardware a také software. V příloze A je na ilustraci popsán životní cyklus hardware a software. Z obrázku jsou patrné některé odlišnosti těchto cyklů.

První etapa životního cyklu koncepce a stanovení požadavků hardwaru a softwaru se příliš neliší. U obou produktů je nutné získat maximum informací o požadavcích a následně je zdokumentovat. Chyby ve specifikaci požadavků jsou velmi častou příčinou poruch softwaru.

V etapě návrh a vývoj již lze spatřit některé odlišnosti. Na základě získaných požadavků je vyhotoven návrh produktu a následně je vyvinut prototyp. U softwaru se

velmi často stává, že vývoj nového softwaru vychází ze základů již funkčního a ověřeného produktu. Stávající produkt je pak víceméně přizpůsoben novým požadavkům. Takováto situace přináší velké finanční úspory. U hardwaru lze tento proces praktikovat jen v omezené míře.

Z pohledu možností testování software je pro účely této práce zajímavá zejména etapa výroby a instalace (v příloze označena jako implementace). V této etapě probíhá nejvíce zkoušek bezporuchovosti softwaru. U hardwaru je často nutné k provádění zkoušek disponovat několika kusy funkčních prototypů. Pokud jsou výsledky zkoušek neuspokojivé, je zapotřebí analyzovat příčinu chyb a určit postup pro jejich opravu. Při testování software se v současnosti často využívá ke zkouškám i návrh architektury. Lze tak předejít některým chybám, které by byly objeveny později. Při testování prototypu software je zpravidla oprava nalezené chyby méně časově a finančně náročnější než u hardware.

Během etapy provozu a údržby mohou být zjištěny závažné chyby produktu. V takovém případě je nutné v co možná nejkratším čase tyto chyby opravit u všech produktů. V případě software jde o poměrně snadný proces. Pokud jsou uživatelé v online spojení s výrobcem, lze všechny instalace aplikace ve velmi krátkém čase aktualizovat. Příkladem v tomto směru mohou být aktualizace operačního systému Windows. Oproti tomu pokud je nalezena chyba například u brzdového systému vozidla, je velmi obtížné informovat o této skutečnosti všechny majitele. Všichni majitelé vozů se pak musí dostavit do určených servisů, kde je porucha opravena. Tento proces je tedy u hardwaru pracnější a finančně náročnější než u softwaru.

Poslední etapa likvidace produktu je u softwaru podstatně jednodušší, než je tomu u hardware. Software lze poměrně snadno odinstalovat ze všech pracovních stanic. Oproti tomu hardware je třeba fyzicky zlikvidovat. Během proces likvidace je však vyprodukován odpad, který je nutné ekologicky zpracovat. Díky tomu jsou náklady na likvidaci mnohem vyšší než u software.

Pokud obsahuje chybu software, může aplikaci využívat např. 100 uživatelů bez sebemenších problémů. U softwaru totiž (mnohem více nežli u hardwaru) záleží jakým způsobem je v provozu využíván. V mnoha výzkumech z posledních let se ukazuje, že většina uživatelů využívá jen cca 30% všech možných funkcí aplikace. Software tak lze v mnoha případech využívat i se známými chybami.

Oproti tomu chyba v hardwaru má okamžitý dopad na všechny uživatele. Zpravidla se pak hardware nesmí dále využívat proto, aby nedošlo k poruše jeho dalších částí a tím i k náročnější opravě.

Ze všech popsaných odlišností životních cyklů vyplývá, že náklady na životní cyklus softwaru jsou mnohem menší ve srovnání s hardware. Také opravy chyb lze u software provádět mnohem flexibilněji. Poruchy v provozu softwaru jsou vždy důsledkem chyb ve specifikaci požadavků (etapa koncepce a stanovení požadavků), návrhu softwaru (etapa návrhu a vývoje) nebo implementaci (etapa výroby a instalace). U hardwaru mohou být poruchy navíc způsobeny využíváním produktu v provozu.

2.5 Dílčí závěr

V této kapitole jsem popsal základní pojmy spojené se spolehlivostí. S těmito pojmy budu pracovat ve zbytku této práce, zejména pak v následující kapitole, která je věnována zkouškám bezporuchovosti. Především v kapitole 5 budu využívat zmíněných ukazatelů vlastností spolehlivosti při popisu realizace návrhu postupu testování softwaru. V této části práce byla vyzdvižena důležitost spolehlivosti produktu. Popsán byl také životní cyklus produktu, ze kterého vychází i životní cyklus softwaru. Na popsané etapy životního cyklu softwaru se budu odvolávat ve zbytku práce. V kapitole 4 pak využiji popsaných modelů životního cyklu software z této kapitoly.

Na konci druhé kapitoly jsou uvedeny některé zásadní rozdíly v životních cyklech hardwaru a softwaru z pohledu spolehlivosti. Ze zmíněného porovnání je zřejmé, že např. náklady na životní cyklus softwaru jsou mnohem menší než v případě softwaru. Tento fakt nejlépe vystihuje etapa likvidace, kde jsou náklady na likvidaci softwaru téměř nulové. Poruchy softwaru na rozdíl od hardwaru nejsou primárně způsobovány využíváním produktu v provozu, ale jsou to důsledky chyb z předchozích etap životního cyklu. Rozdíly v životních cyklech hardwaru a softwaru se odrážejí také ve zkouškách bezporuchovosti. Konkrétní rozdíly ve zkouškách bezporuchovosti jsou pak zmíněny na konci třetí kapitoly.

3 Zkoušky bezporuchovosti

Hlavním cílem zkoušek bezporuchovosti je poskytnout objektivní a reprodukovatelná data o bezporuchovosti objektu [17]. Zkoušky bezporuchovosti softwaru lze provádět prakticky ve všech etapách životního cyklu produktu. Naopak u hardwaru se zkoušky bezporuchovosti provádějí především v etapě instalace a provozu a údržby.

Specifikace zkoušky bezporuchovosti musí dle [4] obsahovat:

- Podmínky při skutečném provozu
- Cíle zkoušek
- Vyjádření účelu zkoušky
- Podmínky výběru zkušebních vzorků a typu zkoušky
- Uspořádání zkoušky
- Sběr a zpracování zkušebních dat
- Vyhodnocení výsledků zkoušky a jejich využití
- Ověření metodiky zkoušky

V oblasti zkoušení bezporuchovosti softwaru jsou všechny výše popsané body definovány v dokumentu plán testování (viz kapitola 4.3.2). Obecně lze tuto specifikaci využít jak u softwaru, tak i hardwaru.

V obecném pojetí zkoušky bezporuchovosti lze podle [17] rozdělit do tří kategorií:

- Určovací – stanovuje odhad hodnoty jednoho nebo více ukazatelů bezporuchovosti, které se mohou použít pro předpověď bezporuchovosti objektů a pro odhad záručních nákladů. Účelem zkoušky je obvykle kvantifikovat bezporuchovost zkoušeného objektu s použitím číselných hodnot jednoho nebo několika klíčových ukazatelů.
- Ověřovací – rozhoduje o shodě hodnoty jednoho nebo více ukazatelů bezporuchovosti uvedených v technických specifikacích se skutečnou hodnotou ukazatelů, dosaženou při zkoušce.

- Srovnávací – stanovuje, zda má objekt A vyšší bezporuchovost než objekt B.

Při provádění zkoušek bezporuchovosti se využívá především kategorie ověřovací. U hardwaru lze využívat všech výše zmíněných kategorií.

Tato kapitola je zaměřena především na zkoušky bezporuchovosti software. Zejména pak na jejich kategorie a úrovně. Pouze v závěru kapitoly je uveden rozdíl mezi zkouškami software a hardware. V praxi se v souvislosti se softwarem často využívá místo pojmu zkouška, pojem test.

3.1 Definice bezporuchovosti

Jak již bylo zmíněno v kapitole 2.2.2, bezporuchovost je dílčí vlastnost, pomocí které se vyjadřuje spolehlivost. Tuto vlastnost lze kvantifikovat pomocí ukazatelů (viz kapitola 2.2.3).

Bezporuchovost (viz kapitola 2.2.2) je definována v [2] jako:

Schopnost objektu plnit požadovanou funkci v daných podmínkách a daném časovém intervalu.

3.2 Kategorie zkoušek bezporuchovosti softwaru

Zkoušky bezporuchovosti u software lze rozdělit do několika kategorií. Tyto kategorie se pak využívají v různých stupních testování. Níže je uvedeno několik základních a často používaných kategorií testů jak jsou popsány v [19]. Mnohdy se využívá spojení více kategorií, jako například u statického testování černé skříňky apod. Níže uvedené rozdělení není jediné možné. Často se lze setkat i s řadou dalších kategorií, které však nemají tak zásadní význam.

Statické a dynamické testy

Podle toho, zda je k testování nutné spuštění software, rozlišujeme statické a dynamické testy. Statické testy nevyžadují běh software. Využívají se tedy zejména v raných fázích životního cyklu software, kdy ještě není vytvořen prototyp. Lze je použít ještě před začátkem psaní kódu na kontrolu specifikace požadavků a analýzu zdrojového kódu. Dynamické testy vyžadují spuštění aplikace. Potřebují proto alespoň spustitelný prototyp software. Používají se v pozdějších fázích vývoje a jsou zaměřeny na provoz software.

Testy bílé skříňky a černé skříňky

Při použití testů černé skříňky není k dispozici přístup k programovému kódu. Software si lze v tomto případě představit jako černou skříňku, jejíž obsah (zdrojový kód) není zvenčí viditelný. Neznáme tedy, jak přesně systém pracuje s daty. Můžeme pouze sledovat, jaký výsledek získáme po vložení vstupních dat. Naopak u testů bílé skříňky máme zdrojový kód k dispozici. Známe tak vnitřní strukturu software. Lépe pak můžeme otestovat pokud možno všechny průchody zdrojovým kódem, zadání neočekávaných vstupních hodnot a další testy, které vycházejí z kontroly zdrojového kódu. Občas se setkáváme s kombinací obou kategorií, ta bývá nazývána jako testy **šedé skříňky**. V praxi se může jednat např. o situaci, kdy software testujeme přes jeho uživatelské rozhraní. Výsledky operací pak ověřujeme pomocí dotazů do databáze.

Funkční a nefunkční testy

Testy, které ověřují, že aplikace správně plní všechny úkoly pro které je určena, nazýváme funkčními. Testují se obecně všechny funkce, které jsou v aplikaci implementovány, a ověřuje se, že fungují správně a že odpovídají požadavkům zákazníka [21]. Ověřuje, zda daný software disponuje funkcemi, které jsou uvedeny v požadavcích zákazníka.

Nefunkční testy spočívají v testování všech vlastností aplikace, které přímo nesouvisí s jejími funkcemi, ale zároveň jsou podstatné pro její správné fungování. Řadí se sem především výkonové testování (Performance testing), které má ověřit například, že aplikace i pod zátěží (větší počet současně pracujících uživatelů, větší objem dat, atd.) bude pracovat dostatečně "svižně". Testuje se také připravenost aplikace pro budoucí nárůst zátěže. A testování neujde ani chování aplikace k počítači, na kterém běží. Tedy třeba to, zda zbytečně nezatěžuje paměť a procesor.

Z výše uvedeného je zřejmé, že funkční testy mají na výslednou bezporuchovost softwaru významný vliv. Proto je na tuto kategorii kladen veliký důraz během celého testovacího cyklu software. Nezáleží přitom na rozsahu testovaného produktu. Nejčastěji se tato kategorie využívá v integrační, systémové a akceptační úrovni testování. Využití této kategorie přináší velké množství odhalených chyb, tedy alespoň ve srovnání s ostatními kategoriemi. Nefunkční testování má také nepochybný vliv na výslednou bezporuchovost software. Bohužel v praxi na něj většinou není příliš

pamatováno nebo se této kategorii testů nedostává dostatečná doba na provedení všech potřebných testů.

Smoke test

Občas se do češtiny nesprávně překládá jako „zahořovací test“. Kategorie Smoke testů se využívá v okamžiku, kdy je dokončen vývoj aplikace a lze ji spustit. Tedy na konci úrovně integračního testování (viz kapitola 3.3). Jedná se o krátký test, který slouží jako rychlé ověření, zda je vyvíjená aplikace připravena pro další fázi testování. Obvykle se provede jen jednoduché ověření, že všechny části aplikace jsou implementovány, nainstalovány a spuštěny. Smoke testy se zaměřují pouze na hlavní funkce programu, které nebývají příliš často upravovány. Často se také kombinují s kategorií testů splněním (viz níže). Jelikož je rozsah smoke testů menší než tomu bývá u ostatních kategorií a pokrývají pouze hlavní funkce, jsou velmi často automatizovány. Pokud nejsou automatizovány v plném rozsahu, zbytek testů je prováděn manuálně.

Úspěšné provedení smoke testů je podmínkou pro vstup do systémové úrovně testování. Proto jsou velmi důležité. U menších projektů, kde je testování software opomíjeno nebo není příliš organizováno, se provádí pouze smoke testy.

Progresní a regresní testy

Progresní testy využíváme při kontrole nových funkcí nebo vlastností aplikace. K jejich správnému provedení je nutná dokumentace, která přesně popisuje nově implementované oblasti. Používáme je ve všech etapách testování.

Regresní testy se využívají při opětovném testování funkcí a vlastností aplikace. Jejich smyslem je ověření, že provedené změny či implementace nových vlastností v aplikaci nemělo žádný vliv na stávající funkce a vlastnosti. Tedy především na oblasti, které zůstaly v programovém kódu nezněměny. Oblasti, které byly předmětem úprav, by správně již měly být otestovány funkčními testy. Takovéto situace z pravidla nastávají po opravení chyb či po novém release[2]. Regresní testy je velmi vhodné automatizovat.

[2] Release je finální verze aplikace, která je připravena pro běžný provoz. Této verzi předchází Debug, který slouží pro odladění chyb.

V praxi jsou regresní testy velmi rozšířené. V různých formách se používají prakticky u většiny projektů. Využívají se hlavně při retestech[3] opravených chyb. Oproti tomu progresní testy nejsou příliš rozšířeny a často je tato kategorie testů rozložena do jiných kategorií.

Testy splněním a selháním

Někdy se uvádí také pojmy pozitivní a negativní testy. U testů splněním jako vstupní hodnoty v aplikaci využíváme jen množinu dat, kterou musí aplikace vždy akceptovat. Kontrolujeme, zda získaný výstup je shodný s výstupem očekávaným v požadavcích od zákazníka.

Při testech selháním do aplikace zadáváme pouze nestandardní data. Naší snahou je aplikaci tzv. "shodit" (způsobit nečekané ukončení běhu programu). Během testů ověřujeme, že výstupní data neobsahují nežádoucí hodnoty, tj. data, která neobsahuje množina očekávaných dat. Obě tyto kategorie se lze využívat v jedné sadě testů. Testy splnění a selhání se mohou prolínat.

Testy splněním a selháním se využívají velmi často. Mnohdy zejména v kombinaci s použitím testů černé skříňky. V praxi se většinou nejprve spustí testy splněním. Pakliže jimi testovaný software projde, lze spustit testy selháním. Nic nám však nebrání v použití obou kategorií najednou, samozřejmě pokud je na to projekt dostatečně připraven (tzn., neobsahuje velké množství chyb, které by odhalily testy splněním). Při testech selháním se doporučují postupy jako např. dělení nulou, vkládání extrémních hodnot apod. Pozor si musíme dávat nejen na bezporuchové chování programu, ale i na správnost chybových hlášení. Oznámení typu: „chyba!“ nebo „operace se nezdařila“ nedávají uživateli moc šancí na zjištění, jaký krok dělá nesprávně. Tato kategorie testů se spouští zejména v systémové úrovni testování. Zde má na celkovou bezporuchovost produktu velký význam. Osobně v praxi kladu velký důraz na testy selháním, protože tyto testy v konečném důsledku velmi zvyšují bezporuchovost software.

[3] Retest je opakování testu. V praxi se tento výraz využívá v situaci, kdy tester nalezne a ohlásí chybu v software, programátor ji opraví a vrátí k následnému retestu. Tedy opětovnému otestování.

3.3 Základní úrovně zkoušek bezporuchovosti softwaru

Základní úrovně testování jsou definovány jako soubor testů, kterými je software testován na daném stupni podrobnosti. Podle toho v jaké fázi a s jakým časovým odstupem od sepsání kódu, se testování provádí, jej dělíme do pěti základních úrovní:

- Testování programátorem (Developer testing)
- Testování jednotek (Unit testing)
- Integrační testování (Integration testing)
- Systémové testování (System testing)
- Akceptační testování (Acceptance testing)

Testování kódu programátorem

Ihned po vytvoření programového kódu, je tento kód prověřen programátorem. Většinou si však programátor netestuje svoji část kódu, ale realizuje se tzv. „test čtyř očí“. To znamená, že kód testuje jiný programátor, než ten který jej napsal. Program je v tomto stupni kontrolován na úrovni zdrojového kódu.

V praxi je bohužel tento stupeň testování často podceňován. Přitom opravy chyby v této části testování software je nejméně nákladná. Sám jsem byl svědkem toho, jak vývojář, aniž by si po sobě jakkoliv zkontroloval kód, předal program k dalšímu testování. Tester se připravil k vlastnímu testování (nastuduje si dokumentaci, připraví testovací data apod.) a začal provádět testy. Záhy však zjistil, že program nelze spustit a nemůže tak ani začít samotné testování. Nahlásil tedy chybu tvůrci programu a ten ji posléze začal opravovat. Tester se tak zbytečně připravoval a začínal provádění testů. Kdyby si po sobě programátor zkontroloval kód dříve, zjistil by chybu mnohem rychleji než tester. Samotná oprava chyby by tak zabrala výrazně méně času. Takovéto postupy zbytečně zabírají čas i ostatním členům týmu.

Testování jednotek

Po ověření kódu programátorem přichází na řadu test jednotek. U objektově orientovaného programování se jedná o testování jednotlivých tříd a metod. Testovanou jednotkou v tomto případě rozumějme samostatně testovatelnou část aplikačního programu. Testy těchto jednotek se zapisují ve formě programového kódu. Proto jej

z pravidla obsluhují vývojáři. Pro vytváření testů se využívá nástrojů na bázi frameworků[4].

Testy jednotek se velmi špatně aplikují na již zaběhlých projektech. U již vytvořených aplikací se většinou musí provést kompletní refaktoring[5] kódu či dokonce mnohem hlubší úpravy. Takováto časová investice se u menších projektů většinou nevyplatí, ale ani u velkých projektů takovýto zásah není příliš šťastný a často se nesetkává s podporou u vedoucího projektu. Proto je vhodné zabývat se těmito testy již v etapě návrhu aplikace a v té době se rozhodnout, zda tyto testy budeme využívat. Jelikož obecně platí, že čím dříve (v rámci životního cyklu software) chybu nalezneme a opravíme, tím méně času nad touto opravou strávíme. Proto bych doporučoval této úrovni věnovat maximální pozornost a to již před samotným vývojem aplikace.

Integrační testování

V době, kdy je vývojář hotov se svými testy, přichází na řadu testovací tým. Integrační testy tedy nepřipravuje programátor, ale především testovací tým. Někdy bývají označovány jako „testy vnitřní integrace“. Musí být ověřena bezchybná komunikace mezi jednotlivými komponentami uvnitř aplikace. Integraci však lze ověřovat nejen mezi komponentami, ale také mezi komponentou a operačním systémem, hardwarem či rozhraním různých systémů. V této fázi se tak testuje integrace dosud jednotlivě ověřených částí. Postupně začínáme testovat integraci mezi dvěma komponentami a postupně přidáváme další. Integrační testy mohou být jak manuální, tak i automatizované.

Úroveň integračního testování je svým způsobem obsažena ve většině testovacích postupů softwaru. U menších projektů je však na tyto testy kladen velmi malý důraz. Má to své logické odůvodnění. Integrační testování lze v testovacím cyklu zcela vynechat. Na výslednou bezporuchovost softwaru to přitom nebude mít žádný vliv, tedy alespoň za předpokladu, že korektně provedeme následující úrovně testování.

---

[4] Framework je prostředí, které slouží pro podporu a organizaci vývoje nové aplikace. Takové aplikace jsou pak napsány za pomoci stejného programového jazyka jako framework. Framework je v podstatě soubor knihoven a kódu uspořádaný tak, aby pokryl co nejvíce funkčních požadavků pro různé aplikace.

Framework má za úkol ušetřit programátorovi čas tak, aby se mohl věnovat jen specifickým požadavkům, které nelze vyřešit za pomocí knihoven z frameworku.

[5] Refaktoring je proces provádění změn v softwarovém systému takovým způsobem, že nemají vliv na vnější chování kódu, ale vylepšují jeho vnitřní strukturu. Je to disciplinovaný způsob pročišťování kódu s minimálním rizikem vnášení chyb (viz [20])

Chyba, kterou bychom odhalili během integračních testů, se zcela jistě projeví v průběhu dalších úrovní testování. Jak již bylo zmíněno dříve, během testování však platí: „čím dříve chybu objevíme, tím méně úsilí nás stojí její oprava“. Proto integrační testy mají svůj význam, nicméně nelze jejich použití nikterak přeceňovat.

Systémové testování

Po ověření správné integrace nastává ten pravý čas na systémové testování. Během těchto testů je aplikace ověřována jako funkční celek. Tyto testy jsou používány v pozdějších fázích vývoje. Ověřují aplikaci z pohledu zákazníka. Podle připravených scénářů se simulují různé kroky, které v praxi mohou nastat. Obvykle probíhají v několika kolech. Nalezené chyby jsou opraveny a v dalších kolech jsou tyto opravy opět otestovány. Součástí této úrovně jsou jak funkční tak nefunkční testy.

Poslední úroveň testů, které se provádějí před předáním produktu zákazníkovi, jsou tedy systémové testy. Tato úroveň testů tak většinou slouží jako výstupní kontrola softwaru. Systémové testování je obsaženo prakticky v každém procesu testování. Bez této úrovně by celé testování softwaru nemělo žádný význam. Bezporuchovost výsledného produktu by byla významně ohrožena. Proto tuto úroveň testů považuji za stěžejní v celém postupu testování software. Na realizaci těchto testů by se mělo myslet již v raném stádiu návrhu postupu testování, tak aby bylo možné obsahu testů co možná nejvíce přizpůsobit očekávanému softwaru.

Akceptační testování

Pokud všechny předchozí etapy testů proběhly bez větších nedostatků, je možné předat aplikaci zákazníkovi. Ten si většinou se svým týmem testerů provede akceptační testy. Ty jsou často prováděny podle připravených scénářů, které společně připravil zákazník s dodavatelem. Testy probíhají na testovacím prostředí u zákazníka. Nalezené nesrovnalosti mezi aplikací a specifikací, jsou reportovány zpět vývojovému týmu. Opravené chyby jsou nasazeny na prostředí u zákazníka.

V této úrovni je zřejmě nejdůležitější, definovat si předem jakou formou bude probíhat oznamování chyb od zákazníka a jak zabezpečit opravení těchto chyb v co možná nejkratší době. Z vlastních zkušeností mohu konstatovat, že zákazník zpravidla očekává určitou chybovost software a je spokojen, když jeho testovací tým nějakou chybu objeví. Velmi nespokojen je však ve chvíli, pokud je takovýchto chyb mnoho nebo jsou to chyby, jež mají zásadní dopad na funkčnost celé aplikace. Pokud jsou již

nějaké chyby v úrovni akceptačních testů objeveny, je nutné v co nejkratším čase tyto chyby opravit a předat zákazníkovi k dalším testům. Velké prodlevy v nalezení a opravení chyby (v akceptační úrovni testování) mohou vést ke zpoždění termínu nasazení softwaru do provozu. Taková situace může mít fatální dopad na úspěch celého projektu.

3.4 Zásady pro provádění zkoušek bezporuchovosti software

Zásadami pro provádění testů v tomto případě rozumějme takové činnosti, jejichž vykonávání má zásadní dopad na průběh a výsledek testovacího cyklu.

Provádění testů bezporuchovosti software je především podmíněno existencí testovacího týmu (testerů). V ideálních případech tento tým disponuje kromě samotných testerů i test manažery (vedoucí testovacího týmu) a architekty (návrháře testů). Máme-li k dispozici tým, můžeme se zaměřit na zásady pro provádění testů software.

R. Patton [19] vyzdvihuje tři nejzásadnější oblasti pro kvalitní provádění testů:

- Plánování testů
- Vlastní testování
- Podávání informací o nalezených chybách

Uvedené termíny jsou však velmi obecné. V následujícím textu podrobněji popisuji, co si pod těmito oblastmi představit.

Plánování testů

Pro provádění testů bezporuchovosti software, musí být vytvořen plán testování (viz kapitola 4.3.2). Obecně lze říci, že tento dokument nám definuje co, jak, kde a kdy testovat. Slouží jako jakási „příručka“ jak postupovat během cyklu testování. V průběhu testů je přípustné některé dříve naplánované činnosti, postupy, harmonogram aj. měnit. Vždy však tyto změny musí být opět zaznamenány do Test plánu. Během celého cyklu testování software je nutné, aby celý projektový tým nepostupoval v rozporu s tímto dokumentem.

Vlastní testování

Máme-li připravený plán testů, můžeme přikročit k vlastnímu testování. Během provádění testů se opět držíme pokynů uvedených v plánu testování. Postupujeme

svědomitě a dle pokynů vedoucího týmu, tak abychom důkladně prověřili celý software a nalezly co největší množství chyb.

Podávání informací o nalezených chybách

Celé snažení testovacího týmu vede k nalezení chyb v kontrolovaném softwaru. Pokud je jakákoliv chyba či nesrovnalost s požadavky nalezena, musí být co nejrychleji zaznamenána a předána odpovědné osobě (viz kapitola 4.5). Odpovědné osoby jsou vždy uvedeny v Test plánu. V oblasti podávání informací se však nejedná jen o včasné informování o nalezených chybách, ale je velmi důležitá i výměna aktuálních informací s ostatními členy, kteří se na projektu podílejí. Z praxe si dovolím odhadnout, že až třetina nahlášených chyb v aplikaci je zaviněna špatnou výměnou informací. Často totiž nastává situace, kdy jsou v průběhu vývoje software upraveny požadavky od zákazníka. Tyto požadavky jsou ihned implementovány do aplikace. Pokud však o této změně není informován testovací tým, nahlásí nové úpravy jako chybu, neboť je vyhodnotí na základě staré specifikace, kterou má k dispozici pro provádění testů. Pokud nedostatek informací nevyústí přímo v chybu, má za následek zbytečné zaneprázdnění pracovníků nebo jiné komplikace v životním cyklu software. Přitom tato činnost vyžaduje minimum úsilí a může zabránit mnoha chybám ať už v samotném produktu či v procesu vývoje, resp. testování.

3.5 Hodnocení zkoušek bezporuchovosti software

Během celého procesu testování software je potřebné kontrolovat a hodnotit aktuální stav projektu. Důvody ke kontrole jsou zřejmé. Nejběžnější důvod bývá kontrola kapacit testerů a programátorů podílejících se na testování projektu. Pokud je chyb mnoho, je dobré doplnit tým o další pracovníky. V opačném případě lze přesunout volné kapacity na jiné projekty.

K průběžnému hodnocení testů se využívá tří základních metrik. Tyto metriky jsou založeny na:

- Chybách
- Testech
- Kódu

Metriky založené na chybách

Chceme-li znát současný stav testů na projektu, můžeme vycházet z aktuálního počtu nalezených chyb. Ovšem samotný údaj o množství nalezených chyb pro nás nemá dostatečný význam (viz například tabulka Tabulka 2:). Jen těžko lze vyvozovat další důsledky z informace o 150 nalezených chybách, měsíc před plánovaným ukončením testů projektu.

Tabulka 2: Příklad prostého údaje o počtu nalezených chyb během testů

Období	Název testovaného softwaru	Celkový počet chyb
20.3.-1.4.2011	Účtenky beta	150

Proto se u metrik založených na chybách, kromě údaje o počtu nalezených chyb, využívá také rozdělení chyb podle funkční oblasti, v které se vyskytují, jejich závažnosti na běh aplikace a stavu jejich opravy. Z těchto informací již lze bezpečně vytvořit sadu užitečných měřítek, jejichž porovnání a poměry jsou dostačující pro hodnocení testů. Příklad takového rozdělení uvádím v tabulce Tabulka 3:.

Tabulka 3: Příklad rozdělení chyb použitím metrik založených na chybách

Funkční oblast	Vysoká závažnost	Střední závažnost	Nízká závažnost	Neopraveno	Celkem
Správa pohledávek	4	12	16	8	32
Tisk a export	0	4	6	4	10
Osobní zprávy	1	9	25	15	35

Hatchensnová [22] uvádí několik užitečných metrik založených na chybách:

$$Bug\ \ fix\ rate = \frac{množství\ opravených\ chyb}{množství\ nalezených\ chyb} * 100[\%]$$

„*Bug fix rate*“ vyjadřuje, kolik procent z celkového počtu nalezených chyb již bylo opraveno. Do čitatele ve vzorci se v praxi dosazuje počet jakkoliv vyřešených chyb. Tedy i například chyby, které byly zamítnuty s odůvodněním, že se jedná o vlastnost aplikace.

$$Efektivnost\ testu = \frac{množství\ nalezených\ chyb\ \ pomocí\ testu}{množství\ chyb\ nalezených\ celkem} * 100[\%]$$

Efektivnost testu určuje, kolik procent chyb bylo nalezeno za pomoci testu. Tento údaj slouží k porovnání jednotlivých testů nebo k porovnání s časem stráveným prováděním testu. Testy, u kterých je zjištěno, že dlouhodobě nepřináší výsledky, jsou nahrazeny jinými.

$$Rychlost\ objevení\ chyby = \frac{množství\ nalezených\ chyb}{doba\ vykonávání\ testu}$$

Rychlost objevení chyby informuje o tom, kolik chyb je průměrně nalezeno za hodinu. Tento údaj nelze přeceňovat. V praxi je zpravidla tato hodnota nejvyšší na počátku testování projektu a postupně klesá s blížícím se koncem testování.

Metriky založené na testech

Aplikace připravenými testy projde za předpokladu, že neobsahuje žádné chyby (po ukončení testů jsou všechny nalezené chyby opraveny) a je připravena k předání do provozu nebo naše testy nedostatečně pokrývají všechny funkce software (v praxi častější případ). Pokud tedy předem známe počet testů (testovacích případů - viz kapitola 4.3.3), kterými musí aplikace během testovacího cyklu projít, lze snadno určit, v jakém stádiu se testování projektu nachází a kolik procent testů již bylo splněno. V praxi se však testy rozdělují do několika kategorií, přičemž všechny nemají stejnou váhu, což musí být v hodnocení zohledněno. Proto tato metrika není vhodná pro všechny kategorie testů. Často se v praxi využívá jen u kategorie funkčních testů. Jako jednotka počtu testů se zde používá testovací případ, případně test suit.

Metriky založené na programovém kódu

Metriky vycházející z programového kódu aplikace, vyjadřují kolik procent tohoto kódu je již otestováno a jakou část ještě zbývá ověřit. Metriky založené na kódu bývají často označovány též jako „pokrytí kódu“ (anglicky code coverage). Podle způsobu, jakým hodnotíme pokrytí kódu, lze metriky rozdělit do několika podtypů:

- Pokrytí příkazů
- Pokrytí hra
- Pokrytí podmínek
- Pokrytí cest

Uvedené podtypy charakterizují z jakého pohledu je na kód pohlíženo. Metriky založené na zdrojovém kódu, se běžně využívají při testování jednotek (Unit test).

3.6 Automatizované zkoušení bezporuchovosti software

Pokud jsou pro daný software vytvořené stabilní manuální testovací případy, které se opakovaně vykonávají, je nasnadě otázka, zda by nebylo možné tyto vykonávat automaticky (tedy bez zásahu lidského faktoru). Oblast automatizace testů je čím dál populárnější a často vyhledávanou možností jak software testovat. Proto jsem se rozhodl se zmínit ve své práci i o možnostech automatizace testů a nastínit tak současné postupy automatizovaného testování software.

V této části podkapitoly bych se rád zmínil o možnosti automatizace provádění zkoušek bezporuchovosti softwaru. Uvedu jejich hlavní význam, realizaci a porovnání výhod a nevýhod jejich použití. Využití tohoto postupu v praxi se budu věnovat v následujících dvou kapitolách.

3.6.1 Význam automatizace zkoušek softwaru

Hlavním cílem automatizace zkoušek bezporuchovosti softwaru je časová úspora při jejich spouštění. Obecně lze říci, že automatizace zkoušek bezporuchovosti softwaru má za účel usnadnit a zefektivnit provádění postupu testování softwaru. Pokud jsou zkoušky prováděny zcela automaticky, tj. bez možnosti zásahu lidského faktoru, výrazně se tím snižuje možnost chybného provedení postupu testování softwaru a tím také pravděpodobnost bezporuchového provozu softwaru. Automatizované zkoušky lze rozdělit do několika kategorií. Tyto kategorie jsou shodné s těmi uvedenými pro manuální zkoušky v kapitole 3.2. Tato skutečnost vyplívá z faktu, že automatizované zkoušky ve většině případů jednoduše zautomatizují provádění manuálních zkoušek.

Použití automatizovaných zkoušek však má své výhody i nevýhody. Porovnání některých pro a proti uvádím v kapitole 3.6.3.

3.6.2 Realizace automatizovaných zkoušek softwaru

Pro realizaci automatizovaných zkoušek, je nutné zajistit některé základní podmínky v postupu testování softwaru. Existující program, který je již v provozu, je obtížné začít automatizovaně testovat. Nejvhodnější etapa pro vytváření těchto zkoušek

je tedy návrh a vývoj softwaru. Architektura aplikace musí umožňovat automatizaci zkoušek, proto je nutné aplikaci s tímto vědomím vytvářet.

Obecně se snažíme pokrýt zkouškami bezporuchovosti co největší část aplikace. U aplikací s krátkou životností (v řádech měsíců, jedna verze) se nám nevyplatí vytvářet tyto testy, naopak u aplikací, kde předpokládáme delší životnost (řádově roky, více verzí), tak tam se nám zcela jistě realizace automatizovaných testů vyplatí. U automatizovaných zkoušek se pokrývají především části, u kterých je reálná hrozba výskytu závažných chyb. Při realizaci testů se však musí přihlédnout k tomu, zda daná část aplikace nebude často upravována. To by totiž znamenalo častou aktualizaci automatizovaných testů.

3.6.3 Výhody a nevýhody automatizace zkoušek softwaru

Výhod pro využití automatizovaných zkoušek bezporuchovosti softwaru je celá řada. Při posuzování výhod a nevýhod jejich použití je vždy nutné vztáhnout dané oblasti na konkrétní projekt. Každý projekt má svá specifika a ty mohou často hrát zásadní roli při možnostech automatizace testů.

Jak již bylo výše zmíněno, asi nejpodstatnější výhodou automatizace zkoušek je časová úspora. Pokud se testy spouštějí a vyhodnocují automaticky, ušetří čas pracovníků, kteří tyto testy dříve manuálně prováděli. Pokud hovoříme o plné automatizaci, lze zkoušky spouštět v kteroukoliv dobu. Běžně se v praxi využívá této skutečnosti a testy se spouští během noci. Jejich spuštění během dne totiž často komplikuje práci ostatním pracovníkům. Pokud člověk prochází neustále ty samé testovací scénáře, často se stává, že některou chybu přehlédne. Využití automatizovaných zkoušek tento nepříznivý jev zcela odstraňuje.

Automatizace zkoušek bezporuchovosti softwaru však sebou nenese jen samé výhody. Jako nevýhodu lze považovat nutnost neustálé údržby zkoušek. Pokud je do aplikace zanesena nějaká změna, automatizované zkoušky musí být ihned aktualizovány. Pokud nejsou ihned aktualizovány, může to mít negativní dopad na střední dobu provozu mezi poruchami. Zkoušky totiž nemusí odhalit všechny chyby a ty tak mohou proniknout i do provozu. Pro provádění automatizovaných zkoušek bezporuchovosti softwaru je nutné vyčlenit speciální prostředí (server) a tomu odpovídající testovací data. To sebou často nese jisté komplikace nebo další časovou investici.

Přes popsané nevýhody si myslím, že je ve většině případů vhodné automatizované zkoušky bezporuchovosti softwaru využít.

3.7 Rozdíly zkoušek pro hardware a software

Pro provádění zkoušek je často nutné produkt uvést do požadovaného stavu. Požadovaným stavem v tomto kontextu chápejme situaci, do které se teoreticky produkt může dostat během svého provozu. V této situaci například na produkt působí nepříznivé vnější vlivy apod. Před prováděním zkoušek bezporuchovosti by správně tyto situace měly být sepsány v dokumentu analýza rizik. Při provádění zkoušek bezporuchovosti u softwaru lze aplikaci do většiny těchto stavů připravit poměrně snadno. V praxi jsem se setkal například se zkouškami softwaru, které měly za úkol ověřit, zda aplikace správně zareaguje na změnu letního času na zimní nebo zda vyhodnotí poruchu datových úložišť. Pro zkoušení hardwaru je však často problematické dosáhnout požadovaných mezních situací. Jako příklad demonstrující problematičnost navození mezních situací u softwaru a hardwaru mohu uvést poruchu některé části hardwaru – jaderného reaktoru elektrárny. Poruchy podobného charakteru v praxi nelze opakovaně nasimulovat, jelikož by mohla ohrozit zdraví pracovníků. Naproti tomu simulování mezních stavů u softwaru představuje podstatně jednodušší skutečnost. Zkoušky bezporuchovosti mezních stavů hardwaru se proto často provádějí za pomoci modelování v softwarech k tomu určených. Provádění zkoušek s využitím modelovacího softwaru nám však nedává stoprocentní jistotu, že hardware takto zareaguje i v praxi.

Zkoušky softwaru jsou méně nákladné a lze s nimi na rozdíl od hardwaru pokrýt prakticky jakékoliv situace, které mohou v provozu nastat.

3.8 Dílčí závěr

Tato kapitola byla věnována zkouškám bezporuchovosti. Především pak zkouškám bezporuchovosti softwaru. Byly zde představeny vybrané kategorie zkoušek a také úrovně, ve kterých se v praxi využívají. Kategorií zkoušek bezporuchovosti softwaru existuje mnohem více, než je uvedeno v této kapitole. Pro potřeby dalších kapitol je však daný výčet dostačující. V krátkosti jsem se zmínil i o možnostech automatizace zkoušek bezporuchovosti. Praktickou ukázku těchto testů uvedu

v následujících kapitolách. Na závěr kapitoly byly zmíněny některé hlavní odlišnosti mezi zkouškami bezporuchovosti hardwaru a softwaru.

Některé kategorie zkoušek využiji ve svém návrhu postupu testování softwaru v následující kapitole. Řídit se pochopitelně budu i zásadami uvedenými v kapitole 3.4. Pro vyhodnocení postupu zkoušení bezporuchovosti softwaru budou poznatky z kapitoly 3.5 využity zejména v kapitole 5.3.

4 Návrh postupu testování software

V této kapitole se budu věnovat všem procesům, které jsou nezbytné pro korektní návrh postupu testování software. Představím zde svůj vlastní návrh postupu testování software a v následující kapitole 5 pak popíši jeho aplikování na konkrétní projekt z praxe. Lépe tak můžeme konfrontovat doposud zmíněné teoretické poznatky z předchozích kapitol s praxí. Návrh postupu testování softwaru uvedený v této kapitole se od jiných návrhů liší především v popisu postupu testování softwaru jako celistvého celku, tedy posloupností všech kroků nutných ke kvalitnímu testování softwaru. Řada návrhů, které jsou obecně prezentovány, jsou soustředěny na konkrétní oblasti. Zde prezentovaný návrh vychází zejména z mých dosavadních zkušeností.

Návrh software zahrnuje plánování a řešení problémů ještě před tím než opravdu nastanou. Stává se tak velmi důležitou (ne-li tou nejdůležitější) oblastí v etapě vývoje software. Navrhování testů je akt systematického promýšlení a plánování řešení před začátkem procesu implementace. Pečlivé plánování a kvalitní návrh zvyšují dlouhodobou hodnotu testů.

Návrh postupu testování softwaru by měl mimo jiné obsahovat i doporučení pro výběr vhodných kategorií a úrovní testování bezporuchovosti. Této problematice jsem se poměrně podrobně věnoval v kapitole 3.2 a 3.3. Výběr vhodných kategorií a úrovní provádění testů vždy velmi úzce souvisí s konkrétním projektem. Nelze tedy navrhnout jaký soubor kategorií či úrovní testů bude vždy optimální používat při realizaci postupu testování softwaru. Přesto si zde dovolím, navrhnou využít jako základní kategorii testování bezporuchovosti softwaru dynamické testování černé skříňky, funkční a smoke testy. Základní úrovně, které by dle mého názoru neměly chybět v žádném postupu testování, jsou testování programátorem a systémové testování. Pokud to finanční situace na konkrétním projektu dovoluje, domnívám se, že realizace výše zmíněných kategorií a úrovní testování bezporuchovosti softwaru, představuje základní kámen úspěšného postupu testování softwaru.

Při správném návrhu testů lze rozhodnout, v jakých případech se vyplatí využít automatizace testů a kde nikoliv. Kvalitní návrh postupu testování a jeho důkladná realizace jsou pilířem úspěšného testování software. Do návrhu postupu testování vstupuje několik faktorů, bohužel ne všechny jsme schopni dopředu identifikovat.

V praxi je nedílnou součástí návrhu postupu testování také odhad času, který zabere příprava a realizace testů. Proces nastavení pracnosti je vždy obtížný. Rozdíl mezi naplánovaným a fakticky stráveným časem testování software se zmenšuje s přibývajícími zkušenostmi test manažera. V některých literaturách (jako například [24]) se uvádí, že se v tomto procesu můžeme orientovat podle doby strávené na implementaci programového kódu. Osobně bych však takový postup příliš nedoporučoval, jelikož dle mých zkušeností čas strávený implementací a testováním software je často velmi rozdílný.

4.1 Vývojový cyklus produktu

Příklady modelů životního cyklu software jsem uvedl v kapitole 2.4.2. Výběr modelu probíhá v první etapě životního cyklu, proto jej nelze měnit v etapě testování ani při návrhu postupu testování. Zvolený model musíme při návrhu postupu testování respektovat a vycházet z něj. Jak jsem již zmínil v kapitole 2.5, pro testování softwaru menších projektů doporučuji využít modely, které jsou založeny na principech Vodopádového modelu. Naopak u větších projektů se rozhodně vyplatí využít modelu RUP. Pokud bych tedy ze své pozice mohl nějakým způsobem ovlivnit výběr modelu životního cyklu vyvíjeného softwaru, řídil bych se podle výše zmíněného. Bohužel jsem se v praxi zatím nesetkal s ochotou naslouchat názorům testovacího týmu při výběru tohoto modelu, který pak svými vlastnosti ovlivňuje také postup testování softwaru.

Navrhuji (pokud to daná situace dovoluje) aktivně se zapojit do výběru modelu životního cyklu softwaru. Výběr modelu zásadně ovlivňuje následný postup testování softwaru, proto si myslím, že by při rozhodování o výběru modelu měly být vyslyšeny i argumenty z oblasti testování softwaru. Mezi hlavní argumenty pro výběr vhodného modelu řadím schopnost modelu flexibilně reagovat na změny v požadavcích na funkce softwaru i během etapy implementace softwaru. Dále bych nedoporučoval využití složitějších modelů (např. RUP) u menších projektů. Výhody plynoucí z propracovanosti složitějších modelů mohou plně projevit jen u větších projektů. Při návrhu postupu testování softwaru navrhuji striktně vycházet z modelu životního cyklu softwaru, který byl zvolen na začátku realizace projektu (viz kapitola 2.4.2). Aby byl postup testování korektní, bez komplikací a jeho výsledky průkazné, musí být postup testování v souladu s tímto modelem. Toho lze docílit pouze za předpokladu, že je vedoucí testovacího týmu (nebo jiná osoba zodpovědná za návrh postupu testování)

podrobně obeznámen s principielním fungováním vybraného modelu životního cyklu, zejména s oblastmi, jež mají přímý vliv na provádění testů bezporuchovosti softwaru.

4.2 Hodnocení rizik na projektu

Obecně je hodnocení rizik velmi obsáhlý obor a je mu v praxi věnována velká pozornost. Pro účely tématu této práce se však zaměřím především na rizika spojená s postupem testování softwaru. V tomto kontextu tedy riziko chápejme jako pravděpodobnost nežádoucí události v součinu s následkem této nežádoucí události. Hodnocením rizik na celém projektu (tedy celého životního cyklu softwaru) se zpravidla zabývá tým analytiků. Tento tým má dostatek informací pro vytvoření kompletní analýzy rizik na celý projekt. Pro návrh postupu testování je však vhodné vytvořit dílčí analýzu rizik pro proces testování softwaru. V takové analýze rizik jsou ohodnocena jednotlivá rizika, jež mohou mít vliv na postup testování. Existence takovéto analýzy v praxi může zamezit řadě negativních událostí v průběhu testování softwaru. Proto bych tuto činnost v návrhu postupu testování rozhodně nepodceňoval a věnoval ji dostatečnou pozornost.

Rizika, která mohou ovlivnit úspěšný proces testování softwaru jsou zaznamenána v dokumentu „Analýza rizik procesu testování softwaru“ (viz kapitola 4.3.1).

Na základě výše zmíněných poznatků navrhuji seznámit se s analýzou rizik na projektu vytvořenou týmem analytiků ihned po jejím vyhotovení. Výsledky této analýzy mohou předejít vážnějším komplikacím v realizaci postupu testování softwaru. V případě nejasností je nutná konzultace s autorem dokumentu. Až po seznámení s tímto dokumentem navrhuji přistoupit k realizaci dokumentu „Analýza rizik procesu testování softwaru“. Jeho obsah je rovněž nutné konzultovat s týmem analytiků a také s vedoucím vývoje. Rozhodně nedoporučuji vytvořit dokument „Analýza rizik procesu testování softwaru“ tzv. “na vlastní pěst“, tj. bez jakékoliv konzultace s vedoucím projektu, analytickým nebo vývojovým týmem. Takové rozhodnutí může mít negativní dopad na samotné testování softwaru, kde by mohla být věnována menší pozornost oblastem nutným ke správné funkci aplikace a naopak.

4.3 Dokumentace nutná pro testování softwaru

Tvorba a následné udržování dokumentace je nedílnou součástí standardního procesu testování software (a pochopitelně ne jen jeho). Kvalitní testování se, dle mého názoru bez připravené dokumentace (alespoň základní) dnes neobejde. Všechny takovéto dokumenty v mnohém usnadňují komunikaci v projektovém týmu a současně se snaží zamezit ztrátám informací např. v případě fluktuace zaměstnanců. Za primární úkol dokumentace osobně považuji vymezení a definování všech postupů pro testování. O formě a druhu dokumentace musím pochopitelně rozhodnout právě v návrhu postupu testování softwaru. Každý dokument bude obsahovat jednoznačný identifikátor. Předejdu tak komplikacím se záměnou dokumentů.

Z výše uvedených důvodů dále navrhuji a níže uvádím minimální rozsah seznamu dokumentů, které podle mne mají zásadní význam pro testování software.

Návrh dokumentů potřebných pro proces testování:

- Analýza rizik procesu testování softwaru
- Plán testování softwaru
- Testovací případ (Test case)
- Testovací scénář (Test scenario)
- Závěrečné hodnocení průběhu testování softwaru

Všechny tyto dokumenty jsou podrobněji popsány v následujících podkapitolách.

4.3.1 Analýza rizik procesu testování softwaru

O hodnocení rizik a analýze rizik na projektu jsem se zmínil již v kapitole 4.2. V této kapitole uvádím konkrétní návrh obsahu zmíněného dokumentu.

Pokud se chceme vyhnout nežádoucím situacím, které mohou mít negativní dopad na proces testování a tím i na bezporuchovost software, musíme si definovat a analyzovat rizika. Proto navrhuji vytvořit dokument analýza rizik procesu testování softwaru, který se skládá ze dvou částí. První část se zabývá riziky procesu testování a druhá pak uvažuje rizika důležitá pro vlastní testování aplikace (tedy vycházející přímo z funkcí testované aplikace). První část může vytvořit analytik nebo testovacím

tým. Myslím si však, že ideální cesta je spolupráce obou těchto týmů, tedy pokud je to v dané situaci možné. První část obsahuje informace o sledované situaci, pravděpodobnost s jakou situace může nastat, případný dopad situace na projekt. Dále podrobnější popis sledované situace a návrh řešení. V tabulce Tabulka 4: je zobrazena ukázka dvou situací, které mohou být popsány v první části navrhovaného dokumentu analýzy rizik. Druhá část dokumentu obsahuje konkrétní informace o funkcích softwaru a je popsána níže.

Tabulka 4: Příklad návrhu analýzy rizik - část proces testování

Situace	**Pravděpodobnost**	**Dopad**	**Popis**	**Návrh řešení**
Nefunkční testovací prostředí	Malá	Kritický	Chybně nastavené prostředí. Nefunkční prostředí, nedostupné pro testery. Nelze na prostředí provádět dané typy testů	Vytvoření a udržování aktuálního kvalitního prostředí. Dostatečné prodlevy mezi nasazováním (release).
Neznalost metodiky testování a testováním celkově	Střední	Kritický	V týmu je špatně pochopen obsah, metody, typy testů.	Nastudování test plánu. Aktualizace Test plánu. Školení ohledně testovacích postupů

Pro samotné testování aplikace je však důležitější druhá část analýzy rizik. Ta obsahuje informace o oblastech aplikace, kde je vysoká pravděpodobnost výskytu chyb a zároveň jaký dopad tyto chyby mohou mít na celkovou funkčnost software. Příklad návrhu takovéto části analýzy je v tabulce Tabulka 5:. Na základě seznamu situací později navrhnu testovací případy (viz kapitola 4.3.3). Za pozornost stojí, že každá situace má své unikátní číslo. Zpravidla každé situaci bude odpovídat minimálně jeden testovací případ, který pokrývá danou oblast aplikace.

Tabulka 5: Příklad návrhu analýzy rizik – část oblastí pro testování softwaru

č.	Kategorie	Situace	Pravděpodobnost	Dopad
1	01 Obecné funkce	Vyhledávání v aplikaci	Střední	Střední
2	01 Obecné funkce	Odkazy osobního menu	Střední	Kritický
3	01 Obecné funkce	Zobrazení zpráv v newsfeedu	Střední	Střední
4	01 Obecné funkce	Notifikace po provedených změnách	Malá	Lehký
5	01 Obecné funkce	Změna jazyka	Střední	Kritický
6	01 Obecné funkce	Zprávy a notifikace přeloženy do vybraného jazyka	Střední	Střední
7	02 Registrace nového uživatele	Úspěšná registrace nového uživatele	Vysoká	Kritický
8	03 Osobní nastavení uživatele	Nastavení uživatele	Střední	Kritický

4.3.2 Plán testování softwaru

Plán testování považuji ve svém návrhu postupu testování softwaru za stěžejní dokument pro celý proces. Při realizaci návrhu jej doporučuji vytvořit jako první dokument před samotným začátkem testování softwaru. Při realizaci postupu testování softwaru musí být jakousi „biblí“ pro každého člena testovacího týmu a musí obsahovat odpovědi na veškeré otázky ohledně procesu testování daného software. Pokud na nějakou otázku z jakýchkoliv důvodů nemůže být odpověď uvedena přímo v dokumentu, bude obsahovat odkazy na místa (nebo osoby), kde se odpověď nalézá.

Návrh plánu testování obsahuje především rozsah postupu testování software, definuje druhy a kategorie testů, stanovuje harmonogram prací.

Navrhuji, aby plán testování minimálně obsahoval:

- **Cíle testování** – definice toho, co očekáváme přesně od provedení testů nad softwarem. Někdy to může být jen ověření nové funkčnosti, jindy naopak kompletní otestování celé aplikace.
- **Seznam plánovaných oblastí k testování** – Přestože by cílem testování při realizaci měly být všechny funkce software, je nutné vytvořit seznam těchto oblastí. K těmto oblastem také navrhuji přiřadit prioritu, která bude určovat, v jakém pořadí budou oblasti otestovány.

- **Kategorie testů** – kategorie testů, které budou využity během testovacího cyklu. Případně jejich využití v různých úrovních testování
- **Seznam testovacích případů** – seznam unikátních identifikátorů testovacích případů. Navrhuji je seřadit podle pořadí, v jakém budou spouštěny
- **Definice rizik pro testování** – definice hlavních rizik, které mohou znemožnit testování. Jejich podrobný seznam s ohodnocením a návrhem řešení těchto situací je zpracován v analýze rizik
- **Požadavky na testovací data** – definice dat, které jsou nezbytná pro provádění testů.

Mimo výše uvedené oblasti navrhuji umístit do plánu testování také harmonogram plánovaných testů. Harmonogram umožňuje čtenářům plánu testování udělat si velmi dobrou představu o tom, jaké činnosti na sebe kdy navazují. Nevýhodou je však situace, kdy se harmonogram mění i během realizace postupu testování softwaru. Udržovat pak dokument stále aktuální může být obtížné. Navrhuji proto poznamenat v plánu testování, že harmonogram je pouze orientační a nelze jej považovat za definitivní. Poznámka proto musí obsahovat sdělení, že je nutné, ověřit si aktuálnost údaje v harmonogramu u vedoucího testovacího oddělení (autora dokumentu).

Dále navrhuji dbát na přehlednost a srozumitelnost dokumentu plánu testování, jelikož je ze své podstaty určen pro všechny pracovníky, kteří se podílejí na vývoji softwaru. Rozhodně nedoporučuji uvádět odborné termíny z oblasti testování softwaru bez detailnějšího popisu.

4.3.3 Testovací případy a scénáře

Testovací případ (Test case)

Během celého testovacího cyklu jsou odhalovány chyby v softwaru. Největší šance na odhalení chyby je využít cílené testování založené na zdokumentovaných testovacích případech.

Testovací případ, často se využívá i anglický výraz „test case“, popisuje konkrétní akce prováděné s určitou softwarovou komponentou a jejich očekávané výsledky [24]. Softwarovou komponentou v tomto případě může být například část

aplikačního rozhraní nebo také softwarový systém běžící na několika strojích souběžně. Testovací případy se vytvářejí jak pro manuální tak i automatizované testy. Pro účely manuálního testování jsou tvořeny seznamem prováděných kroků a očekávaných výsledků. Automatizované testovací případy se také někdy označují jako testovací skript. Tvoří je sada programových instrukcí a na rozdíl od manuálních by měly být schopny sami rozpoznat, zda uspěly či selhaly.

Testovací případ je tedy dokument, popisující určitou činnost, kterou je potřeba otestovat. Obecně lze říci, že obsahuje kroky se skutečnými vstupními hodnotami spolu s očekávanými výsledky.

V tabulce Tabulka 6: navrhuji jednoduchý formulář pro správu testovacího případu. Dále z tohoto důvodu níže uvádím několik podstatných vlastností, které dle mého názoru přispívají ke kvalitě testovacího případu a použiji je i ve svém návrhu testovacího případu.

- **Identifikátor** – jedinečný identifikátor, na který se budu odkazovat z ostatních dokumentů
- **Účel** – vysvětlení k čemu daný testovací případ slouží
- **Podmínky** – seznam potřebných dat či vlivů prostředí podstatných pro provedení testovacího případu
- **Specifikace kroků a vstupů** – seznam všech kroků a souvisejících vstupních dat, nutných pro úspěšné a opakovatelné provedení testovacího případu
- **Očekávané výsledky** – souhrn všech informací, potřebných k určení, zda případ proběhl úspěšně či nikoliv

Tabulka 6: Příklad návrhu jednoduchého testovacího případu

ID:	#1
Název:	Import šeků
Účel:	Ověření zda proběhne import připravených dat
Typ testů:	Verifikační
Čas:	5 min.
Podmínky:	Připravená data (soubory šeků) ve složce pro importy
Kroky:	1. V menu kliknu na odkaz „Import files“ 2. Stisknu tlačítko Import
Očekávaný výsledek	1. Proběhne import dat 2. V kartě „Import files“ se zobrazí informace o posledním importu 3. V kartě „Imported files“ se zobrazí údaje s importovaným souborem 4. V kartě „Update“ se zobrazí nově importované šeky
Provedení testu:	OK
Poznámky:	

U větších projektů se testovacích případů vytváří běžně až statisíce. K jejich správě je pak nezbytné využívat specializovaný systém (u menších projektů se často využívá pouze tabulkový editor). Systémy pro správu testovacích případů (test case manager, zkratka TCM) slouží k vytváření, verzování, uchovávání a spouštění testovacích případů. TCM systémy se v mnoha směrech podobají systémům na evidenci chyb (viz kapitola 4.5). Některé systémy pro sledování chyb umožňují zároveň spravovat jak chyby, tak i testovací případy. Jejich výhodou je i možnost zaznamenávat vazby mezi testovacími případy a v nich nalezenými chybami či sledovat jejich aktuální stav řešení. Proto tento typ systémů doporučuji využít i při realizaci tohoto návrhu. Na jednom projektu však nedoporučuji využívat více TCM. Jak je uvedeno v [24], při použití několika různých TCM během testovacího cyklu nastávají komplikace např. při testech kompatibility, kdy se musí kopírovat testovací případy z jednoho manageru do druhého. Tato činnost sebou kromě zmařeného času přináší i vysoké riziko poškození integrity či obsahu testovacích případů.

Navrhuji, aby testovací případy neobsahovaly zbytečně mnoho informací. V tomto případě doporučuji držet se hesla „méně je někdy více“. S testovacími případy bude v průběhu testování softwaru intenzivně pracovat několik členů testovacího týmu.

Proto vyplňování zbytečných údajů může mít negativní dopad na pracovní morálku a také na dobu nutnou pro provádění testů. Aby mohl být postup testování softwaru efektivní, je zapotřebí pracovat s efektivními testovacími případy. Realizaci návrhu testovacích případů se věnuji níže v kapitole 5.1.4.

Testovací scénář (Test Scenario)

Sada několika testovacích případů tvoří testovací scénář. Testovací případy na sebe musí navazovat a tvořit tak skupinu logicky navazujících kroků, jejichž účelem je otestovat vybranou funkčnost aplikace, např. vytvoření faktury. Pro správu testovacích scénářů lze opět využít tabulkový editor. Tuto variantu navrhuji využít pouze u menších projektů. Testování softwaru u menších projektů se často obejde i bez testovacích scénářů. K vytváření a budoucí správě testovacích scénářů navrhuji využít některou aplikaci přímo k tomu účelu určenou. Tyto aplikace velmi často podporují i správu testovacích případů (např. HP Quality Center). Navrhuji, aby testovacího scénář obsahoval především tyto údaje:

- **Oblasti k testování** - oblasti, vlastnosti a funkce, které se budou testovat v rámci daného scénáře
- **Kategorie testů** - údaj o kategorii testů, do které tento scénář spadá
- **Testovací případy** - seznam všech testovacích případů, které tvoří daný scénář
- **Metriky hodnocení** - metriky pro určení, zda scénář proběhl úspěšně či nikoliv

Přestože testovací scénáře nenajdou mnoho využití u menších projektů, během testování větších projektů své uplatnění rozhodně najdou. Při správě několika stovek testovacích případů se v praxi neobejdeme bez testovacích scénářů a také jejich sofistikované správy. Jak jsem se již zmínil výše, testovací scénáře rozhodně nepatří mezi dokumenty, bez kterých by se nedalo testovat. Záleží jen na rozhodnutí jak kvalitně a přehledně je návrh postupu testování sestaven.

Z tohoto důvodu se domnívám, že pro kvalitní postup testování softwaru je dobré sestavit a využívat testovací scénáře. Usnadňují orientaci v testovacích případech a vytvářejí lepší přehled v oblastech, které jsou pokryty testy. Proto navrhuji vytváření testovacích scénářů jak u velkých, tak i malých projektů. Tvorba a následná údržba těchto dokumentů je časově náročná. Pokud to podmínky projektu (zejména finanční) dovolují, rozhodně doporučuji využít některý z nástrojů pro správu testovacích scénářů

(např. HP Quality Center). Využití specializovaného nástroje přináší kromě jiného úsporu času při tvorbě a správě dokumentů. Využití testovacích scénářů při postupu testování má pozitivní vliv na přehlednost a srozumitelnost souboru všech testovacích případů. Praktickou realizaci výše zmíněného návrhu na využití testovacích scénářů popíši níže v kapitole 5.1.4.

4.3.4 Závěrečné hodnocení průběhu testování softwaru (Test report)

V této souvislosti se v odborné literatuře můžeme také setkat s pojmy jako Test Status Report, Test Summary nebo Test Result. Test report obsahuje zprávu o výsledcích testování software. Zjednodušeně řečeno lze říci, že jde o hmatatelný důkaz o tom, zda byly všechny požadavky implementovány do testované aplikace. Návrh tohoto dokumentu obsahuje souhrnné informace o provedeném cyklu testování. Poskytuje podrobné hodnocení kvality dané verze softwaru, ale také zvoleného postupu testování.

Zákazník ve většině případů vyžaduje při přebírání software i dokument, který popisuje průběh a výsledky testů. K tomu účelu také navrhuji využít test report.

Dokument závěrečného hodnocení navrhuji vytvořit především po dokončení každého testovacího cyklu. V omezeném rozsahu jej navrhuji využít i po dokončení jedné úrovně testů pro čistě interní hodnocení.

Návrh dokumentu závěrečného hodnocení průběhu testování softwaru musí dle mého názoru minimálně obsahovat:

- Seznam testů, které byly provedeny
- Zhodnocení úspěšnosti testů
- Přehled nalezených chyb
- Vyvstalé problémy
- Návrhy na vylepšení testů

4.4 Cíle testování

Stejně jako na začátku každého plánovaného procesu, je potřeba i u procesu testování software, abychom si předem stanovili, jakých cílů chceme postupem testováním dosáhnout. Jinými slovy, abychom mohli své snažení na konci projektu

vyhodnotit. V návrhu si musíme stanovit jasná kritéria, ze kterých lze snadno vyhodnotit míru úspěšnosti provedených testů a celého testovacího cyklu.

Před samotnou realizací testů proto navrhuji důkladně si vyjasnit cíle testování s vedoucím projektu. Výsledek tohoto jednání musí být zaznamenán, nejlépe v plánu testování. Z vlastní zkušenosti doporučuji přikládat k této činnosti maximální důraz. Pokud nejsou cíle testování předem jasně definovány, mohou po dokončení postupu testování nastat komplikace. Nelze totiž objektivně zhodnotit, zda realizovaný postup testování softwaru proběhl úspěšně a korektně či nikoliv. Stanovení cílů postupu testování softwaru na konkrétním projektu popisuji podrobněji v kapitole 5.1.3, kde jsou tyto cíle zaznamenány do dokumentu plán testování.

Na základě výše uvedených zkušeností navrhuji, aby si vedoucí testovacího týmu sjednal osobní schůzku s vedoucím projektu. Tato schůzka musí být první činností v celém postupu testování. Na setkání musí být jasně definovány cíle testování, tedy požadavky od vedoucího projektu na testovací tým, který se bude podílet na bezporuchovosti softwaru. Definici cílů považuji za velmi důležitou. Mimo jiné se od cílů odvíjí i výběr kategorií testů, či použití automatizovaných testů. Navrhuji cíle zaznamenat do plánu testování. Komplikace mohou nastat, pokud se cíle změní během postupu testování. Taková situace může mít negativní dopad úspěšný průběh postupu testování softwaru. Proto navrhuji zřetelně informovat o této možnosti vedoucího projektu během schůzky, kde se definují tyto cíle. Dále doporučuji mezi cíle zařadit přesnou podmínku toho, za jakých okolností lze považovat výsledný postup testování za úspěšný případně neúspěšný. K tomuto účelu je vhodné využít dokument analýza rizik, ze kterého si lze stanovit, jaké oblasti softwaru musí být bezpodmínečně bezporuchové.

Cíle testování softwaru doporučuji mít na paměti po celou dobu postupu testování. Všechny kroky během tohoto procesu musí vést ke splnění těchto cílů.

4.5 Návrh provádění záznamu o chybách

Záznam o chybě (případně poruše) je v podstatě hlavní a nejvíce viditelný výsledek práce testovacího týmu. Proto velmi záleží na formě a obsahu tohoto záznamu (v tom smyslu se často využívá také termín „report chyb“).

Nejednoznačně zaznamenaná chyba bývá mnohokrát předávána mezi testerem a vývojářem. Taková situace může znamenat až desítky promarněných hodin

práce, které pak v konečném důsledku chybí v jiných oblastech. Naopak srozumitelný report usnadní nalezení a včasné opravení chyby. Proto navrhuji řádně proškolit všechny členy testovacího týmu v tomto směru tak, aby se předešlo výše zmíněným nedorozuměním.

Na základě výše zmíněných skutečností, navrhuji držet se při vytváření záznamu o chybách několika základních pravidel:

- Je stručný, výstižný a podrobný
- Popisuje pouze jednu chybu
- Obsahuje přesný postup jak chybu vyvolat, včetně všech známých podmínek
- Definuje očekávaný stav, který ale díky nalezené chybě nelze vyvolat
- Disponuje verzí testované aplikace, použitého operačního systému případně prohlížeče
- Má nastavenu prioritu, které odpovídá dopadu chyby na práci s aplikací (s touto prioritou musí být chyba opravena)
- V příloze je přiložen screenshot (snímek obrazovky s chybou), výpis z logu apod.

Prověřenou cestou, dle mého názoru, jak zajistit kvalitní srozumitelnou formu reportu, je definovat si její přesnou podobu již během přípravy testování. V praxi pak navrhuji určit osobu, která bude za dodržování stanoveného standardu zodpovídat.

S reporty chyb se pak pracuje v programech k tomu vytvořených. Těmto programům na správu reportů chyb se říká bug report systém nebo též bug tracking (např. Mantis, Bugzilla). Osobně se mi velmi osvědčil nástroj Mantis, proto jej navrhuji využít i při realizaci tohoto návrhu.

Po vytvoření a uložení záznamu o chybě začíná její vlastní život. Součástí bug report systému je tedy i možnost nastavit stav ve kterém se momentálně řešení chyby nachází. Často je vedle tohoto stavu i informace o osobě zodpovědné za vyřešení daného stavu.

4.6 Hodnotící kritéria

Kritéria pro hodnocení jsou stanovena v době návrhu postupu testování. Zaznamenány jsou v dokumentu plán testování (viz kapitola 4.3.2). Samotný obsah těchto kritérií často vychází z požadavků na kvalitu od zákazníka. Spíše pro interní hodnocení bych doporučoval využít metriky zmíněné v kapitole 3.5. Rozhodně nedoporučuji využít jako hodnotící kritérium postupu testování množství nalezených chyb. Takováto kritéria nevypovídají o průběhu postupu testování ani o bezporuchovosti softwaru.

Z osobní zkušenosti navrhuji jako kritéria pro hodnocení postupu testování využít dokumentu analýza rizik postupu testování. Jak již bylo výše zmíněno, dokument obsahuje ohodnocení jednotlivých funkčních oblastí aplikace. Lze si tak poměrně snadno stanovit jako hodnotící kritéria například bezporuchovost určité (zpravidla nekritičtější) oblasti funkce softwaru.

4.7 Prostředí pro testy a příprava dat

Pro kvalitní průběh postupu testování softwaru je nutné mít zajištěné stabilní prostředí. Prostředím v daném kontextu chápejme server s nainstalovanou aplikací, která je předmětem testování. Přístup a popis prostředí musí být uveden v plánu testování. V mém návrhu postupu testování softwaru navrhuji zajistit ve spolupráci s vývojovým týmem, včasné spuštění tohoto prostředí. Prostředí by mělo být spuštěno co možná nejdříve. Nejpozději několik dnů před začátkem realizace samotných testů. Prodlevy v tomto směru posouvají celý harmonogram procesu testování, což má pochopitelně dopad buď na kvalitu testování, nebo na harmonogram předání aplikace zákazníkovi.

Společně s funkčním prostředím je nutné připravit kvalitní data, nad kterými budeme testovat aplikaci. Testovací data musí co možná nejvíce odpovídat těm, se kterými bude aplikace pracovat v provozu. Při přípravě testovacích dat opět navrhuji spolupracovat s vývojovým týmem, ale také s týmem analytiků, kteří jsou v kontaktu se zákazníkem. Mají proto detailní informace o požadavcích klienta a také o formátu a obsahu dat, které budou využity v provozu.

Informace o možnostech získání nebo přípravy těchto dat opět musí být uvedeny v plánu testování. Proto se domnívám, že existence testovacího prostředí a vhodných

testovacích dat, je nezbytnou podmínkou pro samotné spouštění testů softwaru. V praxi jsem se také sekal s tvrzením, že testy mohou probíhat na vývojovém prostředí. Tuto variantu rozhodně nedoporučuji. Na vývojovém prostředí neustále probíhá vývoj aplikace a verze softwaru tak nikdy není stabilní. Proto testy které dopoledne proběhnou úspěšně bez nalezení jakékoliv chyby, mohou týž den odpoledne skončit nalezením několika chyb, aniž bychom na první pohled postřehly jakékoliv změny v aplikaci. Software na vývojovém prostředí se upravuje velmi intenzivně a proto získávání aktuálních informací o těchto změnách je zpravidla velmi obtížné. Dále navrhuji využití testovacích dat, která pocházejí přímo od zákazníka. Nebývá vždy jednoduché se k těmto datům dostat. Vlastnoručně vytvořená data však nikdy neprověří chování aplikace tak důkladně, jako právě data, se kterými bude software pracovat v provozu.

4.8 Analýza výsledků testování

Po dokončení testovacího cyklu se vždy hodnotí celý průběh postupu testování softwaru. Dílčí analýzy lze samozřejmě provádět i během procesu testování. Dokument, který navrhuji využít pro analýzu výsledků testování, jsem popsal v kapitole 4.3.4. Při analýze výsledků navrhuji porovnat cíle testování stanovené před začátkem procesu testování (uvedené v plánu testování) s výsledky uvedenými v hodnotícím dokumentu. Takováto analýza výsledků testování by měla sloužit především pro interní účely firmy, která software vyvíjí. Analýza může pomoci při celkovém vyhodnocení postupu testování softwaru. Navrhuji, aby se analýza výsledků zaměřila jednak na úspěšnost provedených testů, výslednou bezporuchovost softwaru, ale také na postup testování jako takový. Tedy například na volbu kategorií testů, kvalitu vytvářených dokumentů apod. Návrh této analýzy musí obsahovat v podstatě všechny oblasti postupu zmíněné v této kapitole.

Proto navrhuji zaznamenávat obecně veškeré informace o průběhu postupu testování softwaru tak, aby bylo možné využít maximum informací při následném sestavování analýzy výsledků testování. Pokud se budeme držet všech návrhů, které jsem uvedl i v předchozích kapitolách, neměl by být se sběrem dat z postupu testování žádný problém. Nepříjemná situace by nastala, pokud by tato data nikde nebyla zaznamenána. Zpětné ověřování relevance dat či jejich původu je mnohdy komplikované a doporučuji této situaci předcházet pomocí zmíněných návrhů. Před každým návrhem postupu testování navrhuji se seznámit s analýzami postupů testování

vyhotovených z podobných projektů tomu našemu. Lze tak lehce předejít chybám nebo komplikacím, které se již podařilo v minulosti objevit na jiných projektech.

4.9 Návrh automatizace manuálních testů

O oblasti automatizace testů softwaru jsem se již zmínil v kapitole 3.6. V této části navrhuji za jakých podmínek automatizaci testů realizovat. Rozhodnutí o realizaci automatizace manuálních testů na projektu by podle mého názoru mělo být podloženo splněním těchto podmínek:

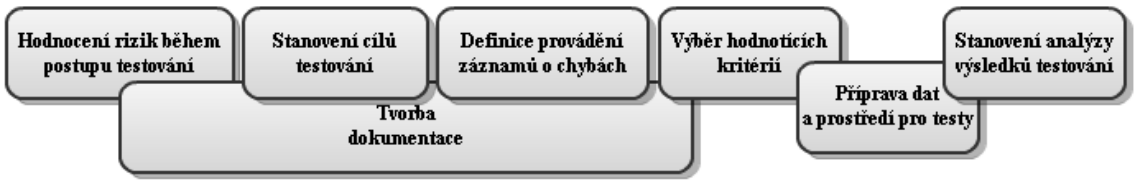
- Na projektu jsou vytvořeny funkční testovací případy (ideálně z kategorie dynamických testů černé skříňky).
- Je plánováno testy softwaru provádět opakovaně (nikoliv tedy pouze jednou, před uvedením do provozu).
- Musí být stanoven dlouhodobý plán vývoje softwaru, který nepočítá s razantní změnou aplikace. Bylo by zbytečné vytvářet automatizované testy pro software, který se brzy razantně změní a testy by se tak musely vytvářet znovu od začátku. Při změnách, které mají zásadní vliv na funkci softwaru zpravidla nelze ani z části využít již dříve vytvořené testy.
- Ke správnému chodu automatizovaných testů je nutné mít připravené testovací data a prostředí, které bude vyhrazeno pouze pro spouštění automatizovaných testů.
- Musí být zjištěny pracovní kapacity pro budoucí údržbu automatizovaných testů a s nimi spojené dokumentace. U neudržovaných testů bez patřičné dokumentace se obecně velmi rychle snižuje efektivnost testu, která nakonec přeroste v chyby v samotných automatizovaných testech.

Při splnění výše popsaných podmínek se domnívám, že je velmi vhodné využít možnosti automatizace testů. Zejména pokud již jsou vytvořeny manuální testy, které tak lze „pouze“ přepsat do nástroje (např. Selenium IDE [25]), který bude testy spouštět. Automatizace testů tak výrazně zvýší efektivitu objevení chyb a přinese další výhody, které byly zmíněny v kapitole 3.6.3.

4.10 Dílčí závěr

V této kapitole jsem představil svůj vlastní návrh postupu testování softwaru. Tento postup lze v praxi realizovat na jakémkoliv projektu testování softwaru. Návrh vychází z informací zmíněných ve třetí kapitole, ale také z mých osobních zkušeností z praxe. Rozhodně se tedy nejedná o jediný možný postup, ale o můj osobní návrh. Proces jakým jsem postupoval při tvorbě návrhu, a který v této podobě v praxi uplatňuji a dále prosazuji, je zobrazen na obrázku Obrázek 6:. Návrh postupuje zleva doprava. Některé činnosti se překrývají, jelikož ovlivňují i ostatní činnosti.

V následující kapitole výše zmíněný návrh aplikuji na konkrétní projekt v praxi. Výsledkem tohoto snažení bude konfrontace mého návrhu s využitím ve skutečném postupu testování softwaru.



Obrázek 6: Schéma procesu návrhu postupu testování

5 Aplikace navrženého postupu v praxi

Zatímco jsem se v úvodních kapitolách věnoval spíše definování pojmů a teoretickému popisu postupu testování softwaru, v této kapitole realizuji návrh postupu testování na reálném projektu v praxi.

Svůj návrh postupu testování software z předchozí kapitoly tedy nyní aplikuji na konkrétní projekt. Tento projekt nazvu „pohledávky“. Jedná se o bankovní aplikaci, která bude spravovat vymáhání pohledávek od klientů banky. Aplikace bude mít webové rozhraní. Zákazníkem je tedy jistá banka, která si u naší firmy objednala software na zakázku. Proběhla důkladná analýza požadavků a ty pak byly zaznamenány do patřičných dokumentů. Na celém projektu se podílí cca 30 zaměstnanců. Dle našich dosavadních měřítek se tedy jedná o malý projekt. Sestaven byl mimo jiné tým testerů. Vedením tohoto týmu jsem byl pověřen já coby vedoucí testovacího týmu (test manažer). Součástí mého týmu se stalo celkem 5 testerů. Z požadavků od klienta vyplynulo, že proces testování softwaru by měl probíhat pouze před uvolněním (předáním) produktu do provozu. O dalším testování během provozu či po případných opravách nebylo rozhodnuto. Tudíž jsem s nimi v mé realizaci návrhu nepočítal. Toto rozhodnutí mělo dopad na některé oblasti realizace návrhu. Zejména pak na tvorbu automatizovaných testů, které jsem se rozhodl vůbec nerealizovat.

5.1 Příprava na testování

Před samotným testováním aplikace bylo nutné získat všechny dostupné informace. V tomto směru mi velmi pomohlo pravidelné setkání s ostatními členy projektového týmu. Dle návrhu bylo nutné ještě před samotnou realizací učinit některé kroky jako je například vytvoření plánu testování či příprava testovacího prostředí. Všechny takovéto činnosti z návrhu jsou popsány v této podkapitole.

5.1.1 Komunikace v týmu

Výměna informací mezi všemi členy projektového týmu je velmi důležitá pro bezproblémový chod celého životního cyklu testování. Zní to jako otřepaná fráze, ale často je tento fakt v praxi podceňován.

Na základě zkušeností ostatních členů z projektu, byla vytvořena pravidelná schůzka zástupců některých týmů, které mají přímý vliv na testování software.

Takovéto jednání jsme nazvali „status testování“. Na schůzku jsem byl přizván již v etapě specifikace požadavků softwaru, tedy cca 2 měsíce před plánovaným začátkem testování. Měl jsem tedy dostatek času a informací pro realizaci navrhovaného postupu testování. Z každé naší schůzky jsem vypracoval stručný výstup, se kterým jsem následně obeznámil ostatní členy testovacího týmu. Každý člen tak dle mého názoru měl dostatek informací o průběhu příprav postupu testování na projektu.

5.1.2 Výběr vhodných typů testů

Dle návrhu výběru vhodných kategorií a úrovní testování, který jsem zmínil v úvodu kapitoly 4, jsem se rozhodl realizovat níže popsané sady testů. Vzhledem k povaze projektu byly vynechány některé úrovně testování, které jsou uvedeny v kapitole 3.3. V přípravě postupu testování jsem počítal s využitím první úrovně, testování programátorem. Úroveň testování jednotek jsem rozhodl vzhledem k tomu, že se jedná o malý projekt nerealizovat. Integrační testování jsem se v omezeném rozsahu rozhodl zachovat. Naopak systémové testování zastává v celém postupu testování softwaru zásadní postavení. Úroveň akceptačních testů si zákazník zajistil sám.

Jako kategorii prováděných testů jsem se rozhodl využít, dle návrhu z kapitoly 4, dynamické funkční testování černé skříňky. Vzhledem k povaze aplikace se mi tato volba jeví jako optimální.

5.1.3 Plán testování

Dle návrhu dokumentu plán testování z kapitoly 4.3.2 byl v přípravě sestaven podle navržených bodů. Na základě jednání s vedoucím projektu byly stanoveny cíle testování. Jako hlavní cíl testování softwaru byla vedoucím stanovena bezporuchovost oblastí označených v druhé části dokumentu analýza rizik, jako nejrizikovější. Pokud tedy v těchto oblastech aplikace nebudou při akceptačních testech u zákazníka nalezeny žádné chyby, proces testování softwaru bude vedoucím projektu považován za úspěšný. Zákazník během sestavování požadavků vytvořil také seznam oblastí a funkcí softwaru, které musí být akceptačními testy potvrzeny jako bezporuchové. Tento seznam byl tedy přidán do dokumentu analýza rizik do kategorie oblastí, které při předání softwaru nesmí obsahovat žádné chyby. Dále si zákazník stanovil podmínku, že ukazatel bezporuchovosti střední doba provozu mezi poruchami musí být prvních 14 dní provozu alespoň 40 hodin. Tyto cíle byly následně zaznamenány do plánu testování. Seznam

plánovaných oblastí pro testování byl obsažen v návrhu plánu testování. Při jeho realizaci jsem však zjistil, že tyto oblasti budou uvedeny také v analýze rizik. Proto jsem se rozhodl je do plánu nezapracovávat a udržovat je aktuální pouze v analýze rizik. S tím souvisí i uvedení seznamu testovacích případů a definice hlavních rizik pro testování, jež měly být také uvedeny v plánu, ale ze stejného důvodu jsem je ponechal pouze v analýze rizik. V plánu testování byl pouze odkaz na tento dokument. Výše zmíněné kategorie testů byly zaznamenány a pro jistotu i stručně popsány do tohoto plánu. Ve spolupráci s analytickým týmem jsem sestavil a zaznamenal požadavky na testovací data. Vzhledem k časovému harmonogramu celého projektu jsem navrhl harmonogram postupu testování a zaznamenal je do plánu testování. Počítal jsem s tím, že by se tento harmonogram mohl časem měnit, ale bylo nutné mít připravené milníky pro správný postup testování softwaru. Tento harmonogram je uveden v tabulce Tabulka 7:.

Tabulka 7: Harmonogram postupu testování softwaru „pohledávky“

Termín	**Plánovaná činnost**
20.-24.9.	Vytvoření **plánu testování**, umístění do sdíleného adresáře.
27.9.-1.10.	Vytvoření dokumentu **analýzy rizik** (za předpokladu, že bude do 29.9. definitivně potvrzen obsah scénářů).
4.-8.10.	1) Do 6.10. musí být zprovozněno **testovací prostředí** – kontrola. 2) Příprava (kontrola, úprava) testovacích dat na testovacím prostředí.
11.-21.10.	Provádění připravených testů softwaru.
28.10. - 2.11.	1) **Nasazení** aplikace na testovací prostředí u zákazníka. 2) **Funkční testy** u zákazníka (pokud bude nasazeno).
3.11. - 17.11.	1) Akceptační testy v bance. 2) Interní **hodnocení plánu** testování - vytvoření reportu.
14.12.	Finální předání produktu zákazníkovi.

Po dokončení byli s plánem testování seznámeni všichni členové projektového týmu. Některé jejich následné připomínky byly do dokumentu zaznamenány. Dokument byl po celou dobu procesu aktualizován a přístupný všem členům týmu.

5.1.4 Testovací scénář

Dle návrhu na sestavení testovacího scénáře v kapitole 4.3.3, jsem se rozhodl realizovat tento návrh i na projektu „pohledávky“. Přesto že projekt „pohledávky“ řadím do kategorie menších projektů, domníval jsem se, že existence testovacích scénářů přispěje ke kvalitnímu a úspěšnému postupu testování softwaru.

Na základě požadavků od klienta byla analytickým týmem vytvořena specifikace požadavků. Aktuální specifikace byla spravována v nástroji Enterprise Architect[6]. S tímto nástrojem jsem zatím nikdy nepracoval. Jelikož jsem měl na přípravu testovací scénářů dostatek času, mohl jsem se věnovat seznámení s tímto nástrojem. Naučil jsem se tedy, jakým způsobem se lze dostat k požadovaným informacím, které potřebuji pro sestavení testovacího scénáře. Byl jsem velmi mile překvapen, když jsem zjistil, že scénáře, podle kterých by mohly testy probíhat, jsou již vypracovány z požadavků klienta. Navíc nástroj Enterprise Architect disponuje poměrně snadnou možností tyto scénáře exportovat. Vytvořil jsem si tedy šablonu, podle níž jsem scénáře, jejichž korektnost byla fakticky potvrzena samotným zákazníkem, vyexportoval do samostatného dokumentu. Šablona obsahovala všechny údaje, které jsem navrhoval v kapitole 4.3.3. Scénáře jsem se však rozhodl rozšířit o záznam data provádění testů a jména pracovníka, který dle daného scénáře testy provedl. Během provádění testů tak budu moci lépe určit, kdo nese odpovědnost za případné pochybení při realizaci testů softwaru. Jelikož jsme ve firmě nedisponovali žádným nástrojem na správu testovacích scénářů, rozhodl jsem se využít pro správu scénářů tabulkový editor. Na malém projektu jako jsou „pohledávky“ by se využití sofistikovanějšího nástroje nevyplatilo jak finančně, tak ani z pohledu úspory času členů testovacího týmu. Součástí exportovaných scénářů byly i testovací případy. Šablonu testovacích případů jsem musel doplnit o identifikátor a účel, tak aby případy obsahovaly všechny vlastnosti, které jsem navrhl v kapitole 4.3.3. Pro správu testovacích případů byl ze stejných

[6] Nástroj Enterprise Architect se používá pro modelování za pomoci UML. Mimo jiné podporuje tvorbu Use Case modelu. Pro export dat je možné si vytvořit vlastní šablonu a výstup uložit do mnoha podporovaných formátů.

důvodů jako u testovacích scénářů využit tabulkový editor. Všechny testovací případy a jejich scénáře tak byly připraveny pro realizaci návrhu postupu testování. Postupně jsem s obsahem výše zmíněných dokumentů začal seznamovat členy testovacího týmu. Testovacích scénářů bylo vytvořeno celkem 8. Testovacích případů, které tyto scénáře obsahovaly, bylo dohromady 35. Tyto dokumenty jsem vyexportoval z výše zmíněné specifikace. Po důkladném seznámení se s obsahem všech těchto dokumentů, jsem se rozhodl nevytvářet další testovací případy či testovací scénáře. Vypracovaná specifikace dle mého názoru pokrývala všechny oblasti a funkce aplikace, které byly uvedeny v dokumentu analýzy rizik a měly být otestovány.

5.1.5 Příprava dat a prostředí

Podle návrhu z předchozí kapitoly jsem ve spolupráci s analytickým týmem zjišťoval, jaká data by byla vhodná pro provádění testů. Nakonec mi byla pro potřeby testování přislíbena data přímo od zákazníka z banky. Problém byl v tom, že data o klientech banky musela být kvůli ochraně osobních dat zašifrována. Původně jsem si myslel, že by to pro provádění testů nemělo znamenat výraznější komplikace. Když jsem však tato data uviděl, zjistil jsem, že jsou pro účely testování aplikace naprosto nepoužitelná. Data totiž byla zašifrována takovým způsobem, že klasické písmena z abecedy byly nahrazeny znaky, jež se běžně vůbec nepoužívají. Využití těchto dat by mimo jiné znamenalo například riziko přehlédnutí některé informace nebo záměny záznamů apod. Proto jsem se raději rozhodl testovací data vytvořit z vlastních zdrojů, tedy za pomoci ostatních členů testovacího týmu.

Přípravu testovacího prostředí jsem diskutoval s vývojovým týmem. Termín zprovoznění tohoto prostředí jsem uvedl také do harmonogramu v plánu testování v kapitole 5.1.3. K tomuto prostředí musí mít přístup všichni členové testovacího týmu, proto jsou přístupová práva uvedena v dokumentu plán testování.

5.1.6 Nástroje testera

Pro testování aplikace „pohledávky“ jsem zvolil několik nutných nástrojů. Jelikož zákazník využívá webový prohlížeč Internet Explorer 6, bylo zřejmé, v jakém prohlížeči budou prováděny připravené testy softwaru. Žádné požadavky na výkon strojů, na kterých aplikace bude v provozu spuštěna, jsem neobdržel. Tudíž tato podmínka ve výběru nástrojů nebyla brána v potaz. Pro správu textových dokumentů

byl zvolen program Microsoft Word, pro analýzu rizik a testovací případy pak tabulkový editor Microsoft Excel. Oznamování a evidence chyb byla dle návrhu realizována v nástroji Mantis.

5.2 Realizace navrženého postupu

Realizace navrženého postupu měla probíhat podle připraveného harmonogramu v kapitole 5.1.3. V době kdy jsem sestavoval tento harmonogram, stále probíhaly změny požadavků od klienta. Této skutečnosti jsem zprvu nepřikládal velkou váhu. Neustálé změny však kromě jiného měly za následek zpoždění ve vývoji připravované aplikace „pohledávky“. Tudíž i samotné provádění testů muselo být nakonec odloženo. O změnách v požadavcích klienta nebyl nikdo z testovacího týmu včas a dostatečně informován. Jak již bylo zmíněno v kapitole 3.4, důsledkem tohoto postupu tak bylo oznamování chyb, které vlastně chybou nebyly.

5.2.1 Oznamování a evidence nalezených chyb

Podle návrhu byl pro oznamování a vlastní evidenci chyb zvolen nástroj Mantis. Tento nástroj je spuštěn na serveru jako webová aplikace. Každá založená chyba může přecházet do různých stavů. Snadno tak lze zjistit, v jakém stavu řešení se právě nachází. Před započetím práce v tomto nástroji je tedy vhodné si stanovit diagram přechodů stavů vložených požadavků nebo chyb. Diagram pro projekt „pohledávky“ je zobrazen v příloze B této práce. V diagramu se vyskytuje pojem „hotfix“, tím chápejme okamžitou opravu poruchy softwaru v provozu. Diagram vytvořil vedoucí vývojového týmu a seznámil s ním ostatní členy, kteří se na projektu podílí. Založen byl tedy v tomto prostředí projekt s názvem pohledávky. Přístup do tohoto projektu byl umožněn všem členům, kteří se podíleli na vývoji produktu. Každý člen se musel přihlásit pod rolí, s jakou vystupuje v životním cyklu softwaru. Vedoucí projektu měl tedy příznak „vedoucí“, členové testovacího týmu pak příznak „tester“ apod. Všechny chyby nalezené v aplikaci taky byly zaznamenány do nástroje Mantis. Příklad takového záznamu uvádím v příloze C.

Během provádění testů byly veškeré chyby a nejasnosti zaznamenávány do nástroje Mantis. Všechny chyby byly postupně dle priorit řešeny vývojovým týmem. Snadné zaznamenávání a správa nalezených chyb se ukázali jako velká přednost

vybraného nástroje. V tomto procesu testování softwaru jsem nezaznamenal žádné komplikace.

Jako metrika pro hodnocení chyb byla vybrána metrika založená na chybách, která je popsána v kapitole 3.5. Souhrn počtu nalezených chyb během realizace návrhu testování softwaru je uveden v tabulce Tabulka 8:. Ve sloupci Testovací prostředí jsou uvedena celkem tři prostředí. Prostředí UAT je testovací prostředí umístěné ve firmě, kde se aplikace vytváří. Na něm se prováděly všechny druhy testů definované v plánu testování. Prostředí FAT je testovací prostředí u zákazníka. Poslední akceptační prostředí je také umístěno u zákazníka, ale probíhají na něm testy řízené pouze zákazníkem. Do této fáze již můj návrh testování nikterak nezasahuje, jelikož bezporuchovost softwaru v dané fázi nikterak nemohu ovlivnit.

Tabulka 8: Souhrn nalezených chyb během realizace postupu testování

Testovací prostředí	Nízká závažnost	Střední závažnost	Vysoká závažnost	Celkem
UAT	38	25	51	114
FAT	31	21	13	65
Akceptační	10	10	4	24

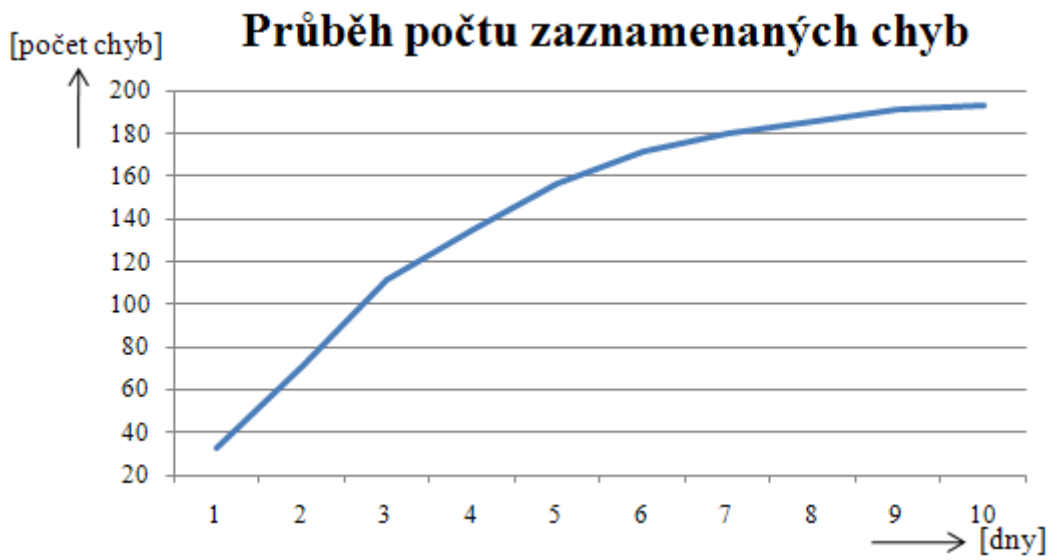
Z tabulky je patrné, že počet chyb nalezených v prostředí FAT je poměrně vysoký (přibližně 33% z celkového množství nalezených chyb). Tato skutečnost byla dána problémy nečekanými problémy během instalace softwaru, o kterých se zmiňuji v následující kapitole. Během akceptačních testů bylo nalezeno celkem 24 chyb, což je přibližně 12% z celkového počtu nalezených chyb. Toto číslo také poměrně vysoké. Vzhledem k faktu, že zákazník předem neměl požadavek na minimální počet chyb, které by toleroval, neznamenal počet těchto nalezených chyb závažnější komplikace pro předání produktu.

Připravené testy byly spouštěny v etapě testování podle harmonogramu celkem 10dní. Během této doby bylo celkem nahlášeno a zaznamenáno 193 chyb. Chyby byly nahlašovány počínaje prvním dnem od zahájení testování aplikace. Průběh počtu těchto záznamů je zobrazen na obrázku Obrázek 7:. V první polovině období vymezeného pro spouštění testů, byl nárůst nových záznamů poměrně strmý. V druhé polovině období již mnoho nových chyb ohlášeno nebylo. Tento průběh ukazuje, že počet nalezených chyb na počátku testování je o mnoho vyšší než na jeho konci. V prvním dni testování

bylo během osmi hodin provozu softwaru nahlášeno celkem 13 poruch. Z níže uvedeného výpočtu vychází hodnota střední doby provozu mezi poruchami 0,62 hodin. Během posledního dne spouštění testů již byly ohlášeny pouze 2 poruchy v osmi hodinovém provozu softwaru. Hodnota střední doby provozu mezi poruchami byla na konci etapy testování 4 hodiny, což je téměř sedmkrát větší hodnota oproti prvnímu dni provádění testů softwaru.

$$\bar{t}_1 = \frac{doba\ \ provozu}{množství\ \ poruch} = \frac{8}{13} = 0{,}62\ hodin$$

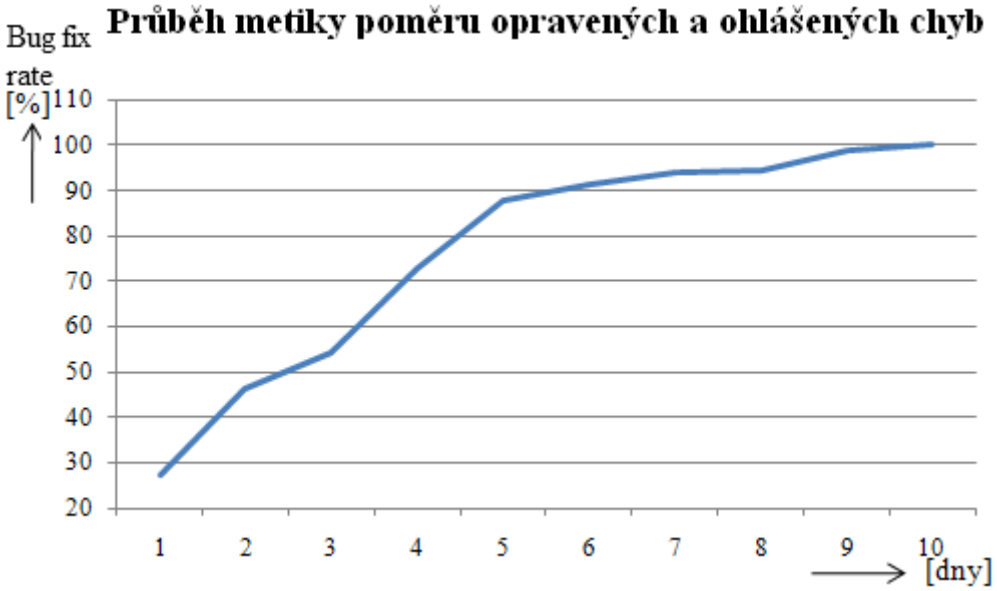
Díky nízké bezporuchovosti softwaru po dokončení vývoje, je pochopitelné větší množství odhalených chyb. Zmíněný průběh jen potvrzuje správnost provedených testů a také dokazuje potřebu testování bezporuchovosti softwaru před jeho uvedením do provozu.



Obrázek 7: Průběh počtu zaznamenaných chyb během procesu testování

Vývojový tým se všech záznamy o chybách snažil vždy velmi rychle vyřešit. Na obrázku Obrázek 8: uvádím vývoj hodnoty metriky Bug fix rate, která vyjadřuje poměr vyřešených a celkově zaznamenaných chyb (podrobněji je popsána v kapitole 3.5.). Během prvních pěti dnů je zřejmý strmý nárůst křivky. Tento jev je dán nárůstem počtu nahlášených chyb, o kterém jsem se zmínil výše. Fakt, že vývojový tým dokázal velmi pružně reagovat na tento vývoj, měl zásadní vliv na úspěšné dokončení testování ve stanoveném termínu. Pokud by nebyly všechny ohlášené záznamy vyřešeny ve stanoveném deseti denním období, znamenalo by to prodloužení doby spouštění testů a tím následné odložení nasazení aplikace na prostředí k zákazníkovi. Během provádění

testů tak byla zachována dostatečná zajištěnost údržby a nevzniklo žádné zpoždění oproti stanovenému harmonogramu průběhu testování softwaru. Danou situaci vyjadřuje vlastnost spolehlivosti Zajištěnost údržby, konkrétně ukazatel Logistické zpoždění. Díky výše zmíněné reakci vývojového týmu tento ukazatel v jednotlivých dnech mohl nabývat pouze hodnot v řádech několika minut, proto nebyl narušen termín předání softwaru zákazníkovi.



Obrázek 8: Vývoj hodnoty metriky Bug fix rate během testování aplikace pohledávky

5.2.2 Řešení nastalých problému

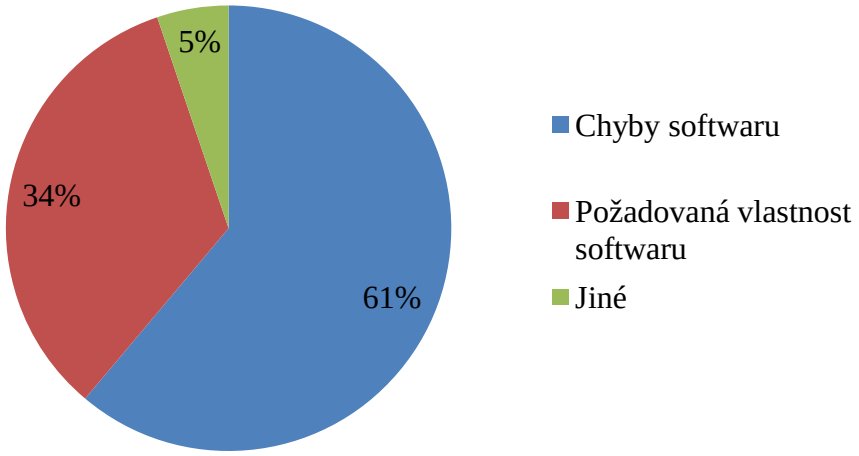
První problém se objevil ještě v průběhu přípravy testů. Vzhledem k neustále se měnícím požadavků ze strany klienta, se začal zpožďovat vývoj připravované aplikace. To mělo pochopitelně dopad i na harmonogram postupu testování. Vlastní testování bylo nakonec odloženo o 14dní. Ovšem o zpoždění bylo rozhodováno prakticky den po dni. Jako vedoucí týmu testerů jsem byl vždy pouze stroze informován o aktuální situaci. Navržený postup testování tak byl dle mého názoru v pořádku, ale byl pouze odsouván díky zpoždění týmu vývoje.

Během postupu testování se objevilo několik nečekaných překážek, na které však nebylo pamatováno v návrhu analýzy rizik. Zřejmě nejvíce času našemu týmu zabraly situace, kdy jsme odhalily nesrovnalosti mezi aplikací a požadavky od klienta. Tyto požadavky se upravovaly i během implementace softwaru. Tento postup není nikterak výjimečný a neměl by nikterak výrazně komplikovat testovací cyklus. Ovšem

pouze za předpokladu, že tým analytiků vždy včas dokument s odsouhlasenými požadavky aktualizuje a poskytne tak potřebné podklady testovacímu týmu. V tomto směru si tedy myslím, že se jednalo spíše o problém na straně týmu analytiků, než ve stanoveném postupu testování. Všechny tyto nesrovnalosti byly ohlášeny a zaznamenávány jako chyby. Později se u některých záznamů ukázalo, že se nejedná o chybu aplikace, ale o požadovanou vlastnost. Všechny zadané záznamy však musely být analyzovány vývojovým týmem, což mělo negativní dopad na hodnotu střední doby opravy. Střední doba opravy byla počítána jako aritmetický průměr všech dob od založení záznamu o chybě až do jejího vyřešení a uvedení opravy na testovací prostředí, během provádění testů na projektu. Během postupu testování srovnatelných projektů, se střední doba opravy pohybovala okolo hodnoty 5 hodin. Díky výše zmíněným problémům byla střední doba opravy na projektu „pohledávky“ přibližně 8 hodin (viz výpočet níže).

$$\overline{t_{OO}} = \frac{doba\ strávená\ opravou\ chyb}{množství\ opravených\ chyb} = \frac{1489}{193} = 7{,}71\ hodin$$

Rozdíl tří hodin způsobily právě situace, kdy se vývojový tým musel zabývat záznamy o chybách, které se nakonec vyhodnotily jako správné vlastnosti funkce, které však bohužel v době nalezení nebyly popsány ve specifikaci aplikace. Přehled konečného vyhodnocení všech ohlášených záznamů o chybách je zobrazen na obrázku Obrázek 9:. Kromě dvou výše zmíněných kategorií jsem zaznamenal i 10 záznamů, které jsem v obrázku označil jako „jiné“. Jedná se o záznamy, které byly označeny jako návrhy na změny chování aplikace a byly dále předány na přezkoumání týmu analytiků nebo se jednalo chybně nastavené testovací prostředí, pracovní stanici apod.



Obrázek 9: Přehled vyhodnocení záznamů o chybách v aplikaci „pohledávky“

Na závěr postupu testování byly naplánovány jakési „předakceptační“ testy u klienta na jeho prostředí. Ovšem vzhledem ke zpoždění vývoje softwaru, nezbylo na tyto testy původně plánovaných 5 dní, ale pouze 2. Další komplikace nastaly při instalaci aplikace na prostředí u klienta. Z různých technický ale i organizačních důvodů se tato instalace protáhla z původně plánovaných dvou dní na celkových 5. Teprve poté mohly přijít na řadu testy u klienta. Během těchto testů bylo nalezeno mnoho chyb způsobených instalací aplikace na nové prostředí. Tyto chyby tak musely být velmi rychle opraveny a implementovány do softwaru. Tento závěrečný proces postupu testování byl bezpochyby nejméně vydařenou částí celé realizace návrhu. V mém návrhu z kapitoly 4 tomuto druhu testů nebyla věnována velká pozornost, proto jejich realizace v praxi neproběhla příliš úspěšně.

5.3 Hodnocení průběhu testování

Po ukončení průběhu všech testů, přichází na řadu závěrečné hodnocení celého průběhu postupu testování. V tuto chvíli je již produkt předán zákazníkovi a ten jej využívá v provozu. Pro interní hodnocení průběhu testování softwaru tak mohu využít i zpětné vazby od zákazníka.

Před spuštěním testů byly stanoveny jejich cíle (viz kapitola 5.1.3). Hlavním cílem byla bezporuchovost vybraných oblastí softwaru. Z akceptačních testů u zákazníka bohužel přišla zpráva o nalezení celkem deseti chyb. Všechny ohlášené chyby byly okamžitě opraveny. Opravy byly implementovány do nové verze

softwaru a ten byl předán zákazníkovi. Nalezené chyby spadaly do oblastí označených v dokumentu analýza rizik jako nejvíce rizikové. Nejednalo se však o oblasti a funkce, na jejichž bezporuchovosti trval zákazník. Průběh testování tak byl interně označen za neúspěšný, naopak z pohledu požadavků klienta byl vnímán jako úspěšný. Díky tomu lze konstatovat, že postup testování nesplnil předem stanovené interní cíle. Naopak z pohledu zákazníka byly požadavky na bezporuchovost předaného softwaru splněny. Bezporuchovost výsledného softwaru tedy byla zákazníkem vnímána kladně. Pro mě coby vedoucího testovacího týmu je to však jasná zpráva, že během provádění testů nebyla věnována dostatečná pozornost plnění interních cílů testování. Bezporuchovost nejrizikovějších oblastí z dokumentu analýza rizik nebyla dostatečně ověřena. Jelikož byly pro provádění testů těchto oblastí vytvořeny všechny potřebné dokumenty a na spouštění testů bylo vyhrazeno dostatek času, domnívám se, že za výše zmíněný neúspěch nese odpovědnost nedůslednost členů testovacího týmu, kteří prováděli testy těchto rizikových oblastí.

5.3.1 Hodnotící dokument

Po dokončení testovacího cyklu, jsem zkompletoval hodnotící dokument, který byl předán zákazníkovi. Tento dokument vycházel z testovacích scénářů a obsahoval počty a druh nalezených chyb.

Pro interní účely jsem vypracoval dokument zaměřující se především na hodnocení zvolených postupů a samotného průběhu testovacího cyklu. K dokumentu se v „připomínkovém řízení“ (forma emailové korespondence), vyjádřili vedoucí všech týmů, které nějakým způsobem zasahovali do procesu testování aplikace. Po zapracování všech připomínek se uskutečnila schůze k diskusi nad průběhem testovacího cyklu, ke které posloužil jako podklad výše zmíněný dokument. Zmíněná diskuse nad zvoleným postupem testování byla součástí celkové diskuse nad kompletním průběhem vývoje aplikace „pohledávky“.

5.3.2 Zapracování nových postřehů

Termínem postřeh v tomto smyslu chápejme subjektivní návrh na zlepšení vyplývající ze zkušeností s proběhnutým procesem testování softwaru. Tyto postřehy zpravidla bývají zaznamenávány během průběhu testování. Po dokončení postupu testování se rozhoduje o možnostech zapracování těchto postřehů. Postřehy z testování

softwaru bych rozdělil na dvě části. První část tvoří postřehy ohledně chování samotné aplikace. Druhou část pak tvoří postřehy související s procesem postupu testování softwaru.

Postřehy k chování aplikace mohou být velmi cenné pro modernizaci softwaru. Testovací tým pracuje s aplikací během provádění testů velmi intenzivně. Jeho připomínky tak jsou velmi pozitivně vnímány zejména týmem analytiků.

Postřehy ke zvolenému postupu testování jsou zaznamenávány během celého procesu postupu testování softwaru. Tyto postřehy přináší kromě testovacího týmu také ostatní členové projektu, kteří se nějakým způsobem podílejí na postupu testování. Jejich připomínky mohou vést ke zlepšení příští realizace postupu.

Po ukončení realizace postupu testování softwaru, byly zpracovány všechny připomínky. Připomínky k chování aplikace byly představeny a vysvětleny týmu analytiků. Poté byly vybrány některé návrhy a předloženy zákazníkovi jako možnost modernizace stávající a již uvolněné verze aplikace. Připomínky k samotnému postupu testování byly projednány na závěrečném setkání členů podílejících se na tomto postupu. Jako příklad těchto připomínek bych uvedl postřeh analytického týmu. Ten připomněl možnost větší spolupráce během provádění testů se členy týmu analytiků. Ti totiž mají přímou zpětnou vazbu od klienta a mají tak velmi dobrý přehled o jeho požadavcích. Výstup z tohoto jednání byl zdokumentován a bude k němu přihlédnuto při příští realizaci návrhu.

5.4 Dílčí závěr

V této závěrečné kapitole byl aplikován návrh postupu testování softwaru, který byl popsán v kapitole 4. Návrh procesu postupu testování byl realizován dle schématu, který je zobrazen v příloze D. Během realizace nastaly komplikace, které jsou zmíněny v kapitole 5.2.2. Většina komplikací byla zapříčiněna zpožděním celého postupu vývoje softwaru. Objevily se však i situace, které nebyly v návrhu postupu vůbec zmíněny. Jednalo se především o testování aplikace na novém prostředí u zákazníka. Na tyto testy díky celkovému posunu harmonogramu nezbyl dostatek času, ale i tak lze tuto situaci vnímat jako komplikaci v postupu testování. Také v plánu testování tyto testy nebyly podrobně popsány. Setkal jsem se také s nesprávným pochopením pojmu systémové testování. Chybou bylo nedostatečné popsání této úrovně testování v plánu testování.

V některých situacích nebyla vyřešena součinnost s vývojovým týmem. Další významný problém představovaly změny v požadavcích klienta ještě během samotné implementace softwaru. S touto situací se opět nepočítalo v dokumentu analýza rizik, a proto nebyla podchycena ani v plánu testování. Tato komplikace nakonec neměla významný vliv na konečnou bezporuchovost softwaru, ale z počátku velmi znepříjemňovala postup testování softwaru a výrazně (celkem o 3 hodiny, viz kapitola 5.2.1) prodloužila střední dobu opravy chyb na hodnotu 8 hodin.

Po předání aplikace zákazníkovi a následnému uvedení softwaru do provozu, bylo během prvních 14 dní objeveno celkem 5 chyb. Střední doba provozu mezi poruchami tak činila 67,2 hodin. Požadavek zákazníka přitom byl minimálně 40 hodin, Tato podmínka tedy byla splněna a zákazník byl s bezporuchovostí softwaru „pohledávky“ spokojen. Díky velmi rychlé opravě (střední doba opravy byla 4 hodiny) výše zmíněných chyb v provozu byla zajištěna dostatečná udržovatelnost a zajištěnost údržby softwaru v provozu. Po uvedení softwaru do provozu byla zákazníkem stanovena střední doba použitelného stavu během dvou týdnů zkušebního provozu stanovena na 295 hodin. Tato hodnota byla představiteli banky vnímána velmi kladně, tudíž pohotovost softwaru byla v provozu dle mého názoru taktéž dostatečná. Ze zmíněných dílčích vlastností spolehlivosti vyplývá, že výsledná spolehlivost softwaru byla dostatečná pro předání produktu zákazníkovi a jeho uvolnění do provozu. Na základě výše zmíněných poznatků si myslím, že realizovaný návrh postupu testování byl správný. Každý projekt má však v praxi svá specifika. Proto je nutné vždy přizpůsobit návrh konkrétnímu projektu.

6 Závěr

Tato práce si kladla za cíl přispět k analýze současných možností testování softwaru. Ve druhé kapitole byly popsány základní pojmy spojené se spolehlivostí. Zvýšená pozornost byla věnována především základním vlastnostem spolehlivosti a jejich vybraným ukazatelům. Tyto termíny pak byly použity v ostatních kapitolách. Ve druhé části kapitoly pak byl uveden životní cyklus produktu včetně popisu jednotlivých etap cyklu. Z životního cyklu produktu vychází i životní cyklus softwaru, který je v kapitole také představen. V druhé kapitole dále byly představeny některé základní modely průběhu životních cyklů softwaru. Kapitolu zakončuje zmínění základních rozdílů mezi životními cykly softwaru a hardwaru.

Třetí kapitola byla věnována zkouškám bezporuchovosti. Představeny byly některé kategorie zkoušek základní bezporuchovosti softwaru a také úrovně, ve kterých se v praxi zkoušky provádějí. Byly zmíněny i některé základní metriky pro hodnocení těchto zkoušek. V krátkosti byla zmíněna možnost automatizace těchto zkoušek. Na závěr kapitoly pak byly porovnány rozdíly zkoušek bezporuchovosti hardwaru a softwaru.

Ve čtvrté kapitole byl představen můj návrh postupu testování softwaru. Návrh byl sestaven na základě informací z předchozích kapitol a také na základě mých dosavadních zkušeností z praxe. Jak se později ukázalo, jeho nevýhodou návrhu byla skutečnost, že návrh nebyl vypracován na konkrétní projekt. Ovšem i s takovou situací se lze v praxi setkat.

V závěrečné kapitole byl realizován návrh postupu testování na konkrétním projektu v praxi. Jak z výsledků vyplynulo, že návrh nepamatoval na některé situace a například plán testování neobsahoval dostatečný popis provádění testů u zákazníka nebo důkladný popis použitých typů testů. Ze závěru páté kapitoly vyplívá, že návrh během realizace vykazoval některé nedostatky. Tyto nedostatky neměly zásadní dopad na bezporuchovost výsledného softwaru. Je však patrné, že návrh postupu testování softwaru nikdy nelze považovat za jediný možný a také jediný správný. Každý projekt má v praxi svá specifika, na které je třeba při návrhu postupu nebo alespoň při realizaci návrhu postupu testování softwaru pamatovat a zohlednit je v tomto procesu.

Cílem práce bylo přispět k analýze současných možností testování softwaru. Zejména díky návrhu a následné realizací postupu testování softwaru byl tento cíl

naplněn. Výsledkem mého snažení je fakt, že navrhovaný postup testování lze v současné době využít, avšak je nutné jej do určité míry přizpůsobit danému projektu.

Literatura

[1] ČSN EN 61508-4. *Funkční bezpečnost elektrických/elektronických/programovatelných elektronických systémů souvisejících s bezpečností - Část 4: Definice a zkratky*. 2002.

[2] ČSN IEC 50 (191) (01 0102). *Mezinárodní elektrotechnický slovník.* 1993.

[3] ČSN ISO/IEC 2382-14 (36 9001). *Informační technologie.* 1999.

[4] ČSN EN 60300-3. *Management spolehlivosti.* 2009.

[5] YANG, Guangbin. *Life cycle reliability engineering*. John Wiley and Sons. 2007. 517 s. ISBN 0471715298

[6] FUCHS, Pavel. *Využití spolehlivosti v provozní praxi.* Liberec: Technická univerzita v Liberci. 2002. 127 s.

[7] ČSN ISO/IEC 12207. *Informační technologie : Procesy v životním cyklu softwaru*. 1997.

[8] CHAPMAN, James. *Software Development Methodology* [online]. Washington, DC. [cit. 2.2.2011]. Dostupný na WWW: <http://www.hyperthot.com/pm_sdm.htm>

[9] ROYCE, Winston. *Managing the Development of Large Software Systems* [online]. Proceedings of IEEE WESCON, 1970. Dostupný na WWW: <http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>

[10] BOEHM, Barry and Papaccio N. Understanding and Controlling Software Costs. In *IEEE Transactions on Software Engineering*. TRW Inc., Redondo Beach, CA. roč. 14, č. 10, 1988, str. 1462-1477. ISSN 0098-5589.

[11] BOEHM, Barry W. A Spiral Model of Software Development and Enhancement. *TRW Defense Syst. Group, Redondo Beach, CA*, roč. 21, č. 5, August 1988. ISSN**:** 0018-9162

[12] BOEHM, Barry W. *Software engineering: Barry W. Boehm's lifetime contributions to software.* Wiley-IEEE Computer Society Press. 2007. 832 s. ISBN: 978-0-470-14873-0

[13] KRUCHTEN, Philippe. *The rational unified process: an introduction*. Addison-Wesley Professional. 2004. 320 s. ISBN 978-0201707106

[14] *Oficiální stránky RUP* [online]. International Business Machines Corp. [cit. 2.4.2011]. Dostupný na WWW: <http://www-306.ibm.com/software/awdtools/rup/index.html>

[15] LEFFINGWELL, Dean. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Addison-Wesley Professional, 2010, 560 s., ISBN 0-321-63584-1

[16] *Rational Unified Process : Best Practices for Software Development Teams*, [online]. Rational Software. [cit. 2.4.2011]. Dostupný na WWW: < http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf>

[17] FOWLER, Martin. *Destilované UML*. Grada Publishing a.s. 2009. 176 s. ISBN 978-80-247-2062-3

[18] HOLUB, Rudolf; VINTR Zdeněk. *Aplikované techniky spolehlivosti : Praktické metody zkoušek spolehlivosti*. Brno: Vojenská akademie v Brně. 2002.

[19] PATTON, Ron. *Testování softwaru.* Praha: Computer Press. 2002. 314 s. ISBN 80-7226-636-5

[20] BECK, Kent. *Refaktoring : zlepšení existujícího kódu.* Grada Publishing a.s. 2003. 394 s. ISBN 8024702991

[21] JORGENSEN, Paul. *Software testing.* Boca Raton: Auerbach. 2008. 416 s. ISBN 0-8493-7475-8

[22] HUTCHESON, Hutcheson. *Software Testing Fundamentals : Methods and Metrics*. Wiley Pub. 2003. 408 s. ISBN 047143020X

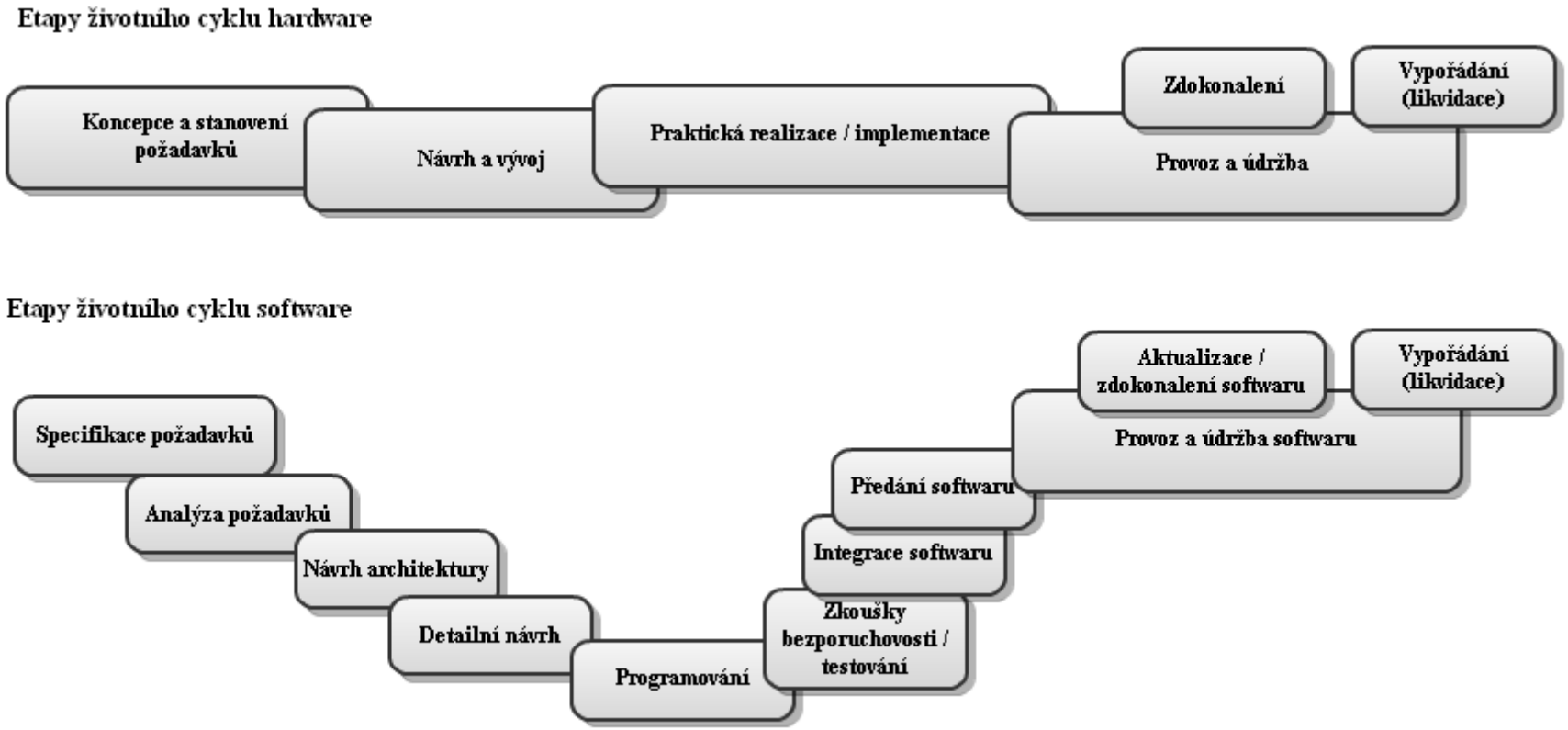
[23] FOURNIER, Greg. *Essential Software Testing*. Boca Raton: CRC Press. 2009. 259 s. ISBN 978-1-4200-8981-3

[24] PAGE, Alan. JOHNSTON, Ken. *Jak testuje software Microsoft.,* Brno: Computer Press. 2009. 384 s. ISBN 978-80-251-2869-5

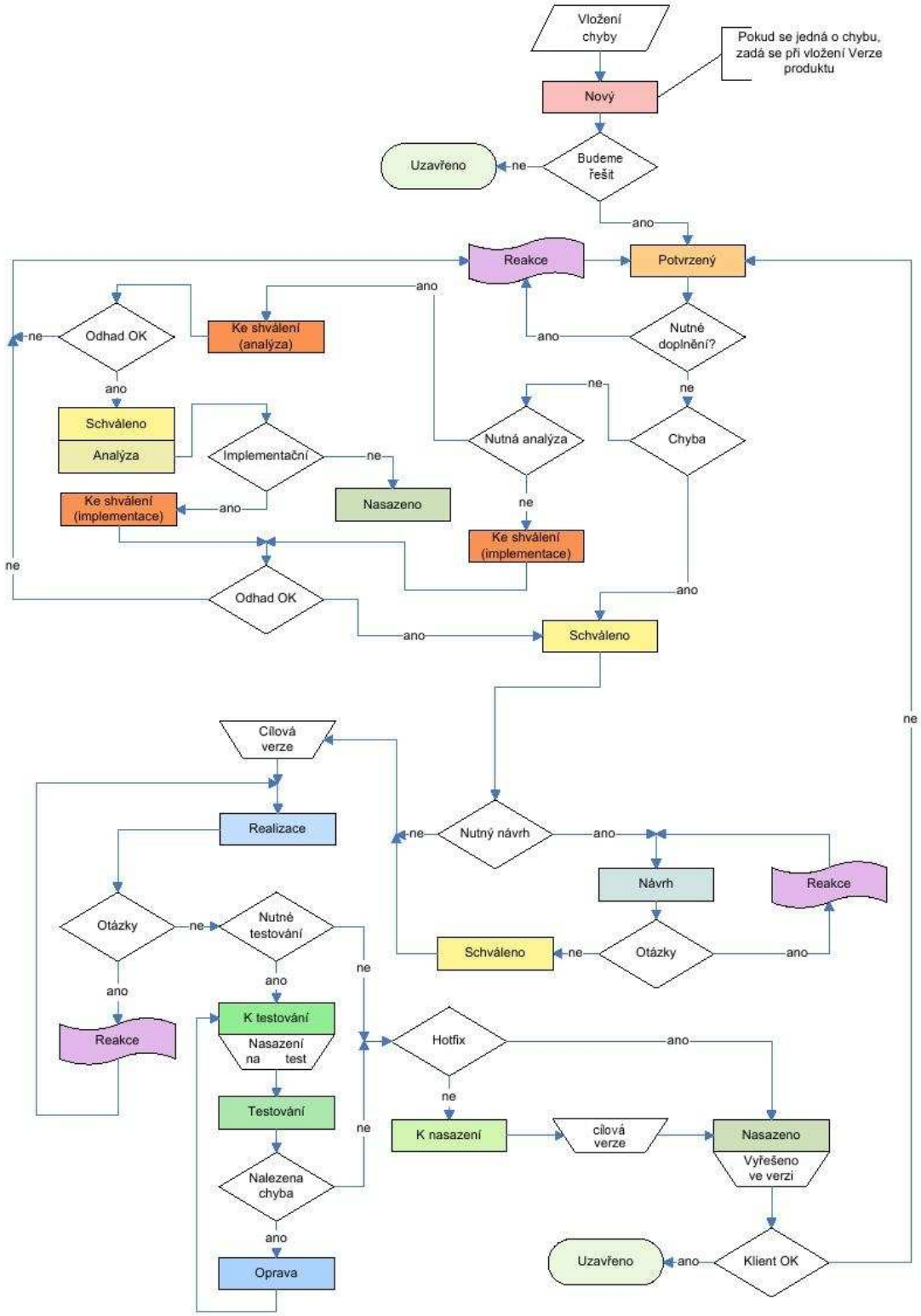
[25] *Oficiální stránky nástroje Selenium* [online]. OpenQA. Dostupný na WWW: < http://seleniumhq.org/>

Přílohy

Příloha A - Etapy životního cyklu hardware a software



Příloha B – Diagram procesu implementace požadavku do softwaru



Příloha C – Příklad záznamu chyby v nástroji Mantis

ID	**Projekt**	**Kategorie**	**Zobrazit stav**	**Datum vložení**	**Poslední změna**
12468	POHLEDÁVKY	Chyba	veřejný	15.2.2010 11:23	31.3.2010 8:32
Reportér	janovsky				
Přiřazen	komar				
Priorita	normální	**Závažnost**	malá	**Reprodukovatelnost**	vždy
Stav	nasazeno	**Řešení**	otevřený		
Platforma	IE 6	**OS**	Windows	**Verze OS**	XP
Verze produktu	1.0.3				
Cílová verze	1.1.0	**Vyřešeno ve verzi**			
Shrnutí	0012468: Při opakovaném založení povinnosti nelze vybrat typ kovenantu				
Popis	Při opakovaném založení povinnosti u jednoho klienta nelze vybrat typ kovenantu.				
Postup:	Vybrán jeden klient z CZ: Lending > otevření detailu klienta > kliknutí na "nová povinnost" > výběr typu povinnosti [kovenant] > zobrazení schovaného comba pro výběru typu kovenantu [+ uskutečnění výběru] > uložení. Otevření detailu klienta > kliknutí na "nová povinnost" > výběr typu povinnosti [kovenant] > nezobrazení schovaného comba pro výběr typu kovenantu!				

Příloha D – Schéma realizace procesu postupu testování softwaru